CA Workload Automation DE

CLI Perspective Help

r11.3 SP2

CA Technologies Product References

This document references the following CA Technologies products:

- CA Process Automation
- CA Spectrum® Service Assurance (CA Spectrum SA)
- CA Workload Automation AE
- CA Workload Automation Agent for Application Services (CA WA Agent for Application Services)
- CA Workload Automation Agent for Databases (CA WA Agent for Databases)
- CA Workload Automation Agent for i5/OS (CA WA Agent for i5/OS)
- CA Workload Automation Agent for Informatica (CA WA Agent for Informatica)
- CA Workload Automation Agent for Linux (CA WA Agent for Linux)
- CA Workload Automation Agent for Micro Focus (CA WA Agent for Micro Focus)
- CA Workload Automation Agent for Microsoft SQL Server (CA WA Agent for Microsoft SQL Server)
- CA Workload Automation Agent for Oracle E-Business Suite (CA WA Agent for Oracle E-Business Suite)
- CA Workload Automation Agent for PeopleSoft (CA WA Agent for PeopleSoft)
- CA Workload Automation Agent for Remote Execution (CA WA Agent for Remote Execution)
- CA Workload Automation Agent for SAP (CA WA Agent for SAP)
- CA Workload Automation Agent for UNIX (CA WA Agent for UNIX)
- CA Workload Automation Agent for Web Services (CA WA Agent for Web Services)
- CA Workload Automation Agent for Windows (CA WA Agent for Windows)
- CA Workload Automation CA 7 Edition
- CA Workload Automation DE
- CA Workload Automation DE Web Client
- CA Workload Automation Desktop Client (CA WA Desktop Client)
- CA Workload Automation ESP Edition
- CA Workload Automation High Availability DE (CA WA High Availability)
- CA Workload Automation Restart Option EE (CA WA Restart Option)
- CA Workload Automation Web Services (CA WA Web Services)
- CA Workload Control Center (CA WCC)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Using the CLI

This section contains the following topics:

Command Line Interface (CLI)

The Command Line Interface (CLI) lets you issue commands against the server and agents. For example, you can issue the STOP command to stop the server. You can also issue common scheduling, operations, and programming commands.

Note: When you issue a command to an agent, a message confirms its submission. However, the command does not provide a message for its success or failure.

The CLI is a perspective of CA WA Desktop Client. You can also install the CLI as a stand-alone utility on your computer. To use the CLI, you must be connected to the server. The commands that you can issue depend on your security permissions.

Note: For more information about using the stand-alone utility, see the *Programming Guide*. For more information about the CLI security permissions, see the *Admin Perspective Help*.

Issue a Command in the CLI Perspective

You can issue commands against the server and agents in the CLI perspective.

Follow these steps:

1. Connect to the server using CA WA Desktop Client.
2. Click the CLI icon on the toolbar.

   The CLI perspective opens.
3. Locate the CLI view and double-click the server that you want to issue commands to.

   A view opens with a Command field and Response area.
4. Enter the command in the Command field.

   Note: For more information about a command, see the corresponding command topic.
5. Click Send.

   The Response area displays a message.

Example: Stop the Server

The following example stops the server:

```
-> stop
(Accepted)
```

help Command—List the Available CLI Commands

You can issue the HELP command to list the available CLI commands.

This command has the following format:

```
help
```

More information:

Issue a Command in the CLI Perspective (see page 10)

help Command—View Help for a Specific Command

You can issue the HELP command to view information about a specific command such as its syntax and operand descriptions.

This command has the following format:

`help command`

command

Specifies the name of the command that you want help for.

Example: View Help for a Specific Command

The following example displays the syntax and operand descriptions for the STOPAGENT command:

```
help stopagent
Usage: STOPAGENT AGENT("agent")

Stop an Agent

Example:
stopagent agent("agent")
stopagent agent("[all]")

agent <string>
        The name of the Agent to stop or [all] for all agents
```

More information:

Issue a Command in the CLI Perspective (see page 10)

Use of Quotes in CLI Commands

All CLI commands follow certain syntax rules. Quotes in CLI have special significance and are an integral part of the syntax. All strings must be enclosed in single or double quotes. You may sometimes have to include quotes within strings.

Note: Booleans and integers are not enclosed in quotes.

Example: PURGECOMPLETEDJOBS Command

The PURGECOMPLETEDJOBS CLI command has two string operands:

```
purgecompletedjobs [olderthan(“olderthan”)] [application(“application”)]
```

The following example explains how the PURGECOMPLETEDJOBS command can be used in a real scenario. The command clears all jobs in the PAYROLL Application that are older than 4 days from the completed jobs repository. Since the *olderthan* and *application* operands are strings, they must be enclosed in quotes.

```
purgecompletedjobs olderthan("now less 4 days") application("PAYROLL")
```

Example: Specifying Quotes Within a String

You can use either single quotes or double quotes for a string, which can be helpful if you want to use quotes within a string, as in the following example:

```
-> updateresourcedefinition name("res") description("This is a 'comment'")
```

Example: Using Backslash as an Escape Character

Alternatively, you can use the backslash as an escape character.

```
updateresourcedefinition name("res") description("This is a \"comment\"")
```

Example: Using Quote Doubling as an Escape Character

You can also use quote doubling as an escape character.

```
-> updateresourcedefinition name("res") description("This is a ""comment""")
```

Examples: Using the Stand-alone CLI from a UNIX shell

These escape sequences are also useful when using the stand-alone CLI from a UNIX shell to provide multiple arguments.

- This example specifies quotes within a string to trigger an Event from a UNIX shell:

  ```
  $ ./cli ejmbp3 9500 schedmaster schedmaster 'triggeradd
  event("CYBERMATION.VERIFY")'
  Event CYBERMATION.VERIFY trigger requested
  ```

- This example uses backslash as an escape character to trigger an Event from a UNIX shell:

  ```
  $ ./cli ejmbp3 9500 schedmaster schedmaster triggeradd
  event\("CYBERMATION.VERIFY"\)
  Event CYBERMATION.VERIFY trigger requested
  ```

Chapter 2: Work with the Server

This section contains the following topics:

about Command—View a List of Users Who Are Connected to the Server

You can view a list of users who are connected to the server by issuing the ABOUT command. The generated list is helpful for administering the server.

This command has the following format:

about

Example: View a List of Users Connected to the Server

This example displays a partial response to the ABOUT command. It lists the CA WA Desktop Client users who are connected to the server.

```
about
Copyright (c) 2008 CA.  All rights reserved.

http://www.ca.com/us/products/product.aspx?id=7833

ESP Server Id: ESP_KOSKA03-TEST_7500
ESP Server Version: 11.1.1.0
Build: 47
Build Date: 20090320
High Availability is NOT Enabled

Connected clients:
 ADMIN@banra04-xp1.ca.com       (155.35.3.241:1465)
 SCHEDMASTER@koska03-winxp.ca.com       (155.35.36.103:1113)
 SCHEDMASTER@padpa02-2k3.ca.com  (155.35.36.158:4604)
```

More information:

Issue a Command in the CLI Perspective (see page 10)

changerole Command—Switch Primary and Standby Roles

In a CA WA High Availability configuration, you can change the server (the Primary or the Standby) that processes workload. You must switch these roles during a manual failback recovery to resume processing on the preferred server. Switching roles is also helpful during server maintenance. For example, suppose that the Primary server requires maintenance. You can switch the roles so that the Standby processes the workload while you administer the Primary server. You can switch Primary and Standby roles by issuing the CHANGEROLE command.

Note: In a CA WA High Availability configuration, changes made to users and permissions take effect immediately on the active server. However, the monitoring server does not become aware of the changes until it is recycled or becomes the active server. For example, when you create a new user, you can use it to log on to the active server, but not the monitoring server. To log on to the monitoring server with the new user, you must recycle the monitoring server or make it the active server by switching roles.

This command has the following format:

```
changerole
```

Note: The Enable failover parameter must be set to true in the Failover tab of the Server Shared Parameters view. Otherwise, the roles are not switched and a message indicating that the CHANGEROLE command is ignored because failover is not enabled.

More information:

Issue a Command in the CLI Perspective (see page 10)

clientsession Command—Ban Users from Opening Client Sessions

You can ban users or groups of users from temporarily opening client sessions to the server by issuing the CLIENTSESSION command.

This command has the following format:

```
clientsession [disconnect|allow] [user("user_ID")|group("group_ID")]
[period(minutes)] [listdisconnected]
```

disconnect

Bans the specified user or group of users from opening client sessions. When a user is disconnected, all the existing client sessions that were opened for that user are closed. Similarly, when disconnecting a group, all the client sessions that were opened by users belonging to the group are closed.

allow

Allows a previously banned user or group of users to open client sessions.

user("*user_ID*")

Specifies the user that is banned from opening client sessions. Alternatively, you can revert the ban of a user by using this operand with the allow operand.

group("*group_ID*")

Specifies the group of users that is banned from opening client sessions. Alternatively, you can revert the ban of a group by using this operand with the allow operand.

period(*minutes*)

Specifies the number of minutes the specified user or group or users is banned for. This operand only applies to the disconnect operand.

Default: 10 minutes

listdisconnected

Lists the currently banned user and groups, together with the ban period.

Notes:

- The disconnect and allow operands are mutually exclusive. If neither of them is specified, the default behavior is disconnect.
- The user and group operands are mutually exclusive.
- If you issue the clientsession command without any arguments, the command lists the currently banned users and groups (listdisconnected operand).
- You cannot disconnect a user or group that has Alter access to the ADMIN.* permission.
- If a user with the ADMIN.* (Alter) permission is a member of a currently banned group, the user will be able to log in.
- If you revert the ban of a user that is a member of a currently banned group, the user will *not* be able to log in.

Example: Ban Members of a Group from Opening Client Sessions

The following example bans members of a group named group1 from opening client sessions:

```
clientsession disconnect group("group1")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

countlist Command—View a List of Artifacts in the System

The server contains different types of artifacts. You can view the total number of each artifact type in your system by issuing the COUNTLIST command.

The CLI lists the following artifact types:

- Agents
- Alerts
- Applications
- Calendars
- Events
- Forecasts
- Groups
- JavaScripts
- Resources
- Users

This command has the following format:

```
countlist
```

Example: View a List of Artifacts in the System

The following example displays a list of artifacts in a system:

```
countlist
Agents:        1
Alerts:        3
Applications: 26
Calendars:     2
Events:       34
Forecasts:     3
Groups:        4
JavaScripts:   8
Resources:     2
Users:         5
```

More information:

Issue a Command in the CLI Perspective (see page 10)

dbinfo Command—Display Server Database Information

You can view information about the relational database configured for your server by issuing the DBINFO command.

This command has the following format:

```
dbinfo
```

Example: Display Server Database Information

The following example displays server database information:

```
dbinfo
DBConnection Pool [Name: RDBMS Manager, DriverClassName:
oracle.jdbc.driver.OracleDriver, URL:
jdbc:oracle:thin:@155.35.40.109:1521:orclcyb, User: system, MaxInPool: 50,
MaxAllowed: 50, In Use: 0, Free: 2, Min: 3, Threshold: 10000, RdmsTpe: Oracle,
ConnectionIdleTimeoutValidateThreshold: 600, ConnectionIdleTimeoutFlushThreshold:
1800]
Available Connections:
SQL Connection Wrapper[usageCount: 144, lastAccessTime: 2007-06-22 18:51:40.906]
SQL Connection Wrapper[usageCount: 19, lastAccessTime: 2007-06-22 18:51:40.906]
```

More information:

Issue a Command in the CLI Perspective (see page 10)

deletestatusmessages Command—Delete the Status Messages from the Dashboard

You can delete all the status messages that display in the Dashboard view of the Monitor perspective if you no longer need the messages. To delete the status messages, issue the DELETESTATUSMESSAGES command.

This command has the following format:

```
deletestatusmessages [threshold("yyyy-mm-dd hh:mm:ss")]
```

threshold("*yyyy-mm-dd hh*:*mm*:*ss*")

(Optional) Deletes the messages older than the specified time.

Note: If you do not specify a threshold, all the historical messages in the Dashboard view of the Monitor perspective are deleted.

Example: Delete the Status Messages Older than a Specified Date

The following example deletes all the status messages from the dashboard that were generated before March 22, 2010 at midnight:

```
deletestatusmessages threshold("2010-04-22 00:00:00")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

disconnectclient Command—Disconnect Inactive Client Connections from the Server

You can disconnect inactive (stale) client connections from the server by issuing the DISCONNECTCLIENT command. You can disconnect stale CA WA Desktop Client, stand-alone CLI, and CA WA Web Client connections.

This command has the following format:

```
disconnectclient id("clientID")
```

id("*clientID*")

Specifies the ID of the client session you want to disconnect. You can use the ABOUT command to retrieve the client ID. The ABOUT command lists all connected client types including CA WA Desktop Client, the stand-alone CLI, and CA WA Web Client.

Notes:

- Exercise caution before disconnecting clients.
- If the client session is inactive, the client session will be disconnected permanently.
- If the client session is active, the client will automatically try to reconnect.

Example: Disconnect a Client Connection

The following example disconnects the yourserver:1977 client connection:

```
disconnectclient id("yourserver:1977")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

encryptpassword Command—Encrypt Passwords for Use in Scripts and Batch Files

If you do not want to use a plain text password in scripts and batch files that use CLI commands, you can issue the ENCRYPTPASSWORD command to encrypt the password. You can then use the encrypted hexadecimal equivalent of the password in scripts and batch files instead of the plain text password.

Note: After encrypting a password, you can connect to the server using the encrypted password or the plain text password. The encrypted password can only be used to issue CLI commands.

This command has the following format:

```
encryptpassword password("password")
```

password("*password*")

Specifies the password to encrypt.

Example: Encrypt Admin Password

The following example encrypts the admin password:

```
encryptpassword password("admin")
0XDBBE4B5982
```

More information:

Issue a Command in the CLI Perspective (see page 10)

gc Command—Perform Java Garbage Collection

To manage memory in your computer, you can release Java memory resources that the server no longer requires. You can perform a Java garbage collection by issuing the GC command. The GC command invokes a Java command that is used to clean memory.

Note: We recommend that you understand the effects of the Java System.gc() command before issuing the CLI GC command. The GC command may not release Java memory resources.

This command has the following format:

```
gc
```

Example: Perform Java Garbage Collection

The following example performs Java garbage collection:

```
gc
```

More information:

Issue a Command in the CLI Perspective (see page 10)

getproperties Command—Display Server Instance Parameters

The server instance parameters contain server-specific and system-specific information, such as ports and server names. You can view these parameters by issuing the GETPROPERTIES command.

This command has the following format:

```
getproperties
```

Example: Display Server Instance Parameters

The following example displays server instance parameters:

```
getproperties
Property: rmi.registry.port=7599
Property: os=Windows NT/2000
Property: failover.preferred.server=true
Property: espresso.local.host=EJMCYB1
Property: clientsession.listenport=7500
Property: manager.inputport=7507
```

More information:

Issue a Command in the CLI Perspective (see page 10)

getproperty Command—Display a Server Property

When the server starts, its JVM (Java Virtual Machine) properties are logged in a tracelog.txt file. You can display these properties by issuing the GETPROPERTY command.

This command has the following format:

```
getproperty [property("property")]
```

property("*property*")

(Optional) Specifies the property in the tracelog.txt file to display.

Default: All properties

Example: Display a Server Property

The following example displays a server property named java.version:

```
getproperty property("java.version")
Property: java.version=1.5.0_08
```

More information:

Issue a Command in the CLI Perspective (see page 10)

licensestatus Command—View License Status

You can issue the LICENSESTATUS command to view the total number of licenses available, the number of licenses in use, and the temporary license's expiration date.

This command has the following format:

```
licensestatus
```

Example: View License Status

The following example displays details about a license status:

```
licensestatus
The license will expire on Jul 7, 2007.
Users of Agent:  (Total of 100 licenses available)
Agent:  1 licenses checked out
Users of VirtualAgent:  (Total of 100 licenses available)
VirtualAgent:  0 licenses checked out
```

More information:

Issue a Command in the CLI Perspective (see page 10)

listquiescestate Command—Display the Server Quiesce State

You can determine whether the server is quiesced by issuing the LISTQUIESCESTATE command.

This command has the following format:

```
listquiescestate
```

More information:

Issue a Command in the CLI Perspective (see page 10)

memcheck Command—Check Server Memory Usage

You can check memory usage information for the server by issuing the MEMCHECK command. This command displays the total free memory and maximum heap size.

This command has the following format:

```
memcheck
```

Example: Check Server Memory Usage

The following example displays memory usage information of the server:

```
memcheck
Using 1.04% of total memory. Used: 10.59 Allocated: 37.95 MB. Free: 1005.53 MB. Max
Heap: 1016.13 MB.
```

More information:

Issue a Command in the CLI Perspective (see page 10)

movehistorydata Command—Move History Data to Stage Tables

The server collects information about completed Applications and stores this information in the relational database. Over time, the history data can become huge. To create more disk space for the database and improve server performance, you can issue the MOVEHISTORYDATA command to move some of the history information in the database tables to stage tables.

Note: When the data has been moved, a message is added to the tracelog.txt file indicating the number of Applications and rows that were moved. For more information about the trace log, see the *Admin Perspective Help.*

This command has the following format:

```
movehistorydata [olderthan("olderthan")]
```

olderthan("*olderthan*")

Moves history data older than the specified time to stage tables.

Note: The value must be a valid schedule criteria statement that resolves to a date and time. For more information about a valid schedule criteria statement, see the *Define Perspective Help*.

Example: Move History Data Older than a Specified Date

The following example moves all the history data that is older than a month to stage tables:

```
movehistorydata olderthan("now less 1 month")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

purgecompletedjobs Command—Clear the Server Completed Jobs Repository

When the server runs, it collects information about active and completed jobs. Your server relational database stores this information. The server completed jobs repository stores information related to completed jobs. To maintain performance, you must periodically clear the server completed jobs repository by issuing the PURGECOMPLETEDJOBS command.

Notes:

- This command is used for CA WA Desktop Client tables and does not affect job history.
- We recommand that you clear the server completed jobs repository after 10,000 jobs have been completed.

This command has the following format:

```
purgecompletedjobs [olderthan("olderthan")] [application("application")]
```

olderthan("*olderthan*")

(Optional) Clears completed jobs older than a scheduled time.

application("*application*")

(Optional) Clears completed jobs that belong to an Application. You can specify a generation number to clear completed jobs for a specific Application generation and its previous generations.

Note: You can use the asterisk (*) as a wildcard for zero or more characters and the question mark (?) as a wildcard for a single character.

Examples:

- application(“PAYROLL”) clears all completed jobs that belong to Application PAYROLL.
- application(“PAYROLL.5”) clears all completed jobs in generation 5 of Application PAYROLL and its previous generations.
- application(“APPL*”) clears all completed jobs that belong to Applications with names that start with APPL.

Example: Clear Completed Jobs in Certain Generations of an Application

The following example clears all the completed jobs that are older than two hours in generation 5 of Application APPLICATION_1 and its previous generations from the server:

```
purgecompletedjobs olderthan("now less two hours") application("APPLICATION_1.5")
```

Example: Clear Completed Jobs Belonging to Certain Applications Using a Wildcard

The following example clears all the completed jobs that are older than four days and belong to Applications with names that start with T (for example, TEST) from the server:

```
purgecompletedjobs olderthan("now less 4 days") application("T*")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

quiesce Command—Quiesce a Server

You can quiesce (pause) the server by issuing the QUIESCE command. Quiescing the server prevents the server from temporarily submitting jobs or triggering workload while the server remains active. Jobs that are running when the server is quiesced continue to run to completion. After you quiesce a server, you can selectively unquiesce certain Application generations and Events, giving you more control over what runs.

This command has the following format:

```
quiesce
```

More information:

Issue a Command in the CLI Perspective (see page 10)

resetgen Command—Reset Application Generations

You can reset one or all Applications by issuing the RESETGEN command. This command resets the Application's generation count to zero and purges its jobs.

This command has the following format:

```
resetgen [application("application")]
```

application("*application*")

(Optional) Identifies the name of the Application you want to reset. You can specify application("all") or omit this operand to reset all Applications.

Default: All Applications

Examples: "PAYROLL", "all"

Note: The server returns an error if the Application is active (not completed). Complete the Application and re-enter the command. If the Application is complete, the command resets the Application generation count to 0.

Example: Reset the Application Generation

The following example resets the generation count of the Application APPL1 to zero and purges its jobs:

```
resetgen application("APPL1")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

stop Command—Stop the Server

You can stop the server by issuing the STOP command.

This command has the following format:

```
stop
```

Example: Stop the Server

The following example stops the server:

```
stop
```

More information:

Issue a Command in the CLI Perspective (see page 10)

threadcount Command—Display the Number of Threads Used by the Server

You can view the number of threads currently running on the server by issuing the THREADCOUNT command.

This command has the following format:

```
threadcount
```

Example: Display the Number of Threads Used by the Server

The following example displays the number of threads currently running on the server:

```
threadcount
 [Thread group=system;active groups=6;active threads=62]
 [Thread group=main;active groups=4;active threads=51]
 [Thread group=RESOURCESERVER;active groups=0;active threads=3]
 [Thread group=SCHEDULER;active groups=0;active threads=2]
 [Thread group= SERVER_7500;active groups=0;active threads=13]
 [Thread group=WORKSTATION_SVR;active groups=0;active threads=23]
 [Thread group=RMI Runtime;active groups=0;active threads=2]
```

More information:

Issue a Command in the CLI Perspective (see page 10)

threaddump Command—Display the Current Threads Stacktrace

You can view the current threads stacktrace used by the server. You can view the stacktrace by issuing the THREADDUMP command.

This command has the following format:

```
threaddump
```

Example: Display the Current Threads Stacktrace

This example displays a partial response to the THREADDUMP command. It displays the current threads stacktrace used by the server.

```
threaddump
 [Thread group=system;active threads=57]
    Reference Handler
        java.lang.Object.wait(Native Method)
        java.lang.Object.wait(Unknown Source)
        java.lang.ref.Reference$ReferenceHandler.run(Unknown Source)
    Finalizer
        java.lang.Object.wait(Native Method)
        java.lang.ref.ReferenceQueue.remove(Unknown Source)
        java.lang.ref.ReferenceQueue.remove(Unknown Source)
        java.lang.ref.Finalizer$FinalizerThread.run(Unknown Source)
```

More information:

Issue a Command in the CLI Perspective (see page 10)

threadlist Command—Display the Current Threads Used by the Server

You can view the current threads used by the server. You can view the threads by issuing the THREADLIST command. This command does not display the stacktrace of the current threads.

This command has the following format:

```
threadlist [cpu(true|false)]
```

cpu(true|false)

(Optional) Specifies whether to sort the list based on CPU usage. Options are the following:

true

Sorts the list based on CPU usage.

false

Does not sort the list based on CPU usage.

Default: cpu(false)

Example: Display the Current Threads Used by the Server

This example displays a partial response to the THREADLIST command. It lists the current threads used by the server.

```
threadlist
 [Thread group=system;active threads=57]
    Reference Handler [0.0 Seconds ]
    Finalizer [0.15 Seconds ]
    Signal Dispatcher [0.0 Seconds ]
    RMI TCP Accept-7599 [0.0 Seconds ]
    RMI TCP Accept-0 [0.0 Seconds ]
    RMI Reaper [0.0 Seconds ]
    GC Daemon [0.15 Seconds ]
    RMI TCP Accept-0 [0.0 Seconds ]
    RMI LeaseChecker [0.0 Seconds ]
```

More information:

Issue a Command in the CLI Perspective (see page 10)

unquiesce Command—Unquiesce a Server

You can unquiesce the server by issuing the UNQUIESCE command to release a quiesced server. The server can then resume submitting jobs and triggering workload. You can also use this command to unquiesce specific Application generations or Events.

This command has the following format:

```
unquiesce [appl("application.generation")] [event("eventprefix.eventname")]
```

appl("*application.generation*")

(Optional) Specifies the name of the Application generation that you want to unquiesce.

application

Specifies the Application name.

generation

Specifies the generation number.

Example: APPL("appl.1,appl.2")

Note: You can also use a wildcard for a partial name. For example, APPL* unquiesces all generations of Applications with names that start with APPL. You can specify multiple Application generations separated by commas or specify *.* to unquiesce all active Application generations.

event("*event*")

(Optional) Specifies the name of the Event that you want to unquiesce.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

Example: EVENT("CYBER.Event1,CYBER.Event2")

Note: You can use a wildcard for a partial name. For example, CYB* unquiesces all Events with prefixes that start with CYB. You can specify multiple Events separated by commas or specify *.* to unquiesce all Events.

Example: Unquiesce an Application Generation

The following example unquiesces the Application ALFSDF.1:

```
unquiesce appl("ALFSDF.1")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

uptime Command—Display How Long the Server has Been Running

You can view how long the server has been running by issuing the UPTIME command.

This command has the following format:

```
uptime
```

Example: Display How Long the Server has Been Running

The following example displays the amount of time the server has been running:

```
uptime
1 Hour 40 Minutes 9.554 Seconds
```

More information:

Issue a Command in the CLI Perspective (see page 10)

Chapter 3: Work with Agents

This section contains the following topics:

countagents Command—Display the Number of Agents that Meet Criteria

You can display the total number of agents defined in the Topology. You can limit the agents counted by status (active, inactive, quiesced) or by agent type. To display the number of agents, issue the COUNTAGENTS command.

This command has the following format:

```
countagents [status("status"] [type("type")]
```

status(“*status*”)

(Optional) Limits the agents counted by status. Options are active, inactive, and quiesced.

type(“*type*”)

(Optional) Limits the agents counted by agent type. Options are the following:

- appserv—Application Services Agent
- db—Database Agent
- informatica—Informatica
- microfocus—Micro Focus Agent
- mssqlserver—Microsoft SQL Server
- oracle—Oracle E-Business Suite Agent
- os400—OS/400 Agent
- peoplesoft—PeopleSoft Agent
- remoteexecution—Remote Execution
- sapr3—SAP-R/3 Agent
- tandem—Tandem Agent
- unix—UNIX Agent
- virtual—Virtual Agent
- vms—Open-VMS Agent
- winnt—Windows Agent
- ws—Web Services Agent
- zos—z/OS Agent

Examples: Display the Number of Agents that Meet Criteria

- The following example displays the total number of active Windows agents:

```
countagents status("active") type("winnt")
1
```

- The following example displays the total number of inactive agents:

```
countagents status("inactive")
3
```

More information:

Issue a Command in the CLI Perspective (see page 10)

createagent Command—Add an Agent to the Topology

You can add an agent to the Topology to work with the server using the CREATEAGENT command.

This command has the following format:

```
createagent name("name") [description("description")] template("template")
[container("container"] address("address") [port(port)] [charcode("charcode")]
[version("version")] [encryptkey("encryptkey")] [serveralias("serveralias")]
[fromagentencryptkey("fromagentencryptkey")] [snmpenabled(true|false)]
[heartbeatfrequency(heartbeatfrequency)] [heartbeatmaxretry(heartbeatmaxretry)]
[persistmgrchange(true|false)] [encryptionmethod("encryptionmethod")]
[user("user")] [userids(userid(id("id") password("password"))...)]
```

name("*name*")

Specifies the name of the agent to be added.

Limits: Up to 16 characters

Notes:

- The server changes the agent name to upper case. Verify that the agentname parameter in the agentparm.txt file is in upper case; otherwise, the server and the agent cannot communicate.
- An agent cannot have the same name as a user defined in the Security view.
- An agent cannot have the same name as an existing agent.

description("*description*")

(Optional) Defines a description of the agent.

template("*template*")

Identifies the type of agent to be added. Options are the following:

- appserv—Application Services
- db—Database
- informatica—Informatica
- microfocus—Micro Focus
- mssqlserver—Microsoft SQL Server
- oracle—Oracle E-Business Suite
- os400—i5/OS
- peoplesoft—PeopleSoft
- remoteexecution—Remote Execution
- sapr3—SAP-R/3
- tandem—Tandem (virtual)
- unix—UNIX
- vms—Open-VMS (virtual)
- winnt—Windows
- ws—Web Services
- zos—z/OS

Note: Before adding a Tandem or Open-VMS virtual agent, you require an agent that acts as a parent for the virtual agent. The parent agent runs on UNIX, Linux, or Windows and connects to the virtual agent using Telnet.

container("*container*")

(Tandem and Open-VMS virtual agents only) Specifies the name of the real (parent) agent that the virtual agent belongs to. It is required to add a virtual agent to the Topology.

address("*address*")

Specifies the IP address or DNS name of the computer where the agent is installed.

port(*port*)

(Optional) Specifies the port number that the agent uses to listen for traffic. The port number is not supported on virtual agents.

Default: 7520

Limits: 1024-65534

Note: This port number must match the communication.inputport parameter in the agentparm.txt file.

charcode("*charcode*")

(Optional) Specifies the character set. The character set is not supported on virtual agents. Options are ASCII and EBCDIC.

Default: EBCDIC (z/OS agent); ASCII (all other non-virtual agents)

version("*version*")

Identifies the release number of the agent. The release number is not supported on virtual agents.

Note: The value depends on the agent type. You can find the list of values in the Release number drop-down list when you add an agent to the Topology in CA WA Desktop Client.

encryptkey("*encryptkey*")

Specifies the encryption key the agent uses to communicate with CA Workload Automation DE. The encryption key is not supported on virtual agents.

Notes:

- CA Workload Automation DE and the agent must have the same encryption key to communicate. The agent's encryption key is stored in a text file (encrypted). The path to that file is set by the security.cryptkey parameter in the agentparm.txt file. If the keys are different, the agent and CA Workload Automation DE cannot communicate and an AGENTDOWN state occurs when you try to run workload.
- If you specify NONE in the encryptionmethod operand, set this value to NOENCRYPTION.

serveralias("*serveralias*")

(z/OS agents only) Specifies the name the agent uses to communicate with the server. It is required to add a z/OS agent to the Topology.

Limits: Up to 16 characters

fromagentencryptkey("*fromagentencryptkey*")

(z/OS agents only) Specifies the encryption key used from agent to server. It is defined for the COMMCHAN initialization parameter in the agent definition data set. It is required to add a z/OS agent to the Topology.

snmpenabled(true|false)

(Optional) Indicates whether SNMP is enabled. This option is not supported on virtual agents.

Default: false

Note: If you specify true, you need to update the SNMP information in the agentparm.txt file.

heartbeatfrequency(*heartbeatfrequency*)

(Optional) Specifies the frequency you want the server to send the heartbeat signal to in minutes. This option is not supported on virtual agents.

Default: 5

Limits: 0 and above

Note: If you want individual agents to have their own heartbeat frequencies, you can set the internal shared parameter named Global agent heartbeat interval in minutes to 0 (zero).

heartbeatmaxretry(*heartbeatmaxretry*)

(Optional) Specifies the number of heartbeat signals the server attempts before it sends an SNMP message indicating agent inactivity. This option is not supported on virtual agents.

Default: 1

Limits: 1 and above

persistmgrchange(true|false)

(Optional) Indicates whether the changes to server properties that affect agent communication are permanently changed on the agent. If you change the ID, address, or port of the server in the Topology, the server sends the agent the updated server connection information at the next heartbeat signal, allowing the server and agent to communicate. Options are as follows:

true

Updates the communication parameters in the agentparm.txt file with changes made to the corresponding properties in the server Topology. For example, if you change the server address and port in the Topology, the communication.manageraddress_*n* and communication.managerport_*n* parameters in the agentparm.txt file are updated with the new server address and port.

false

Updates the server connection information every time the agent is restarted, allowing the server and agent to communicate for that session only. The communication changes are not saved in the agentparm.txt file.

Note: This option only applies to Release 7 agents and higher and is not supported on virtual agents.

encryptionmethod("*encryptionmethod*")

Specifies the encryption method the CA Workload Automation DE uses to encrypt messages. The encryption method is not supported on virtual agents. Options are the following:

DES

Specifies Data Encryption Standard encryption. It uses a 56-bit key. Encryption key length: 56 bits (16 hexadecimal characters).

DESEDE

Specifies 3DES encryption. It uses the DES algorithm in EDE (encrypt-decrypt-encrypt) mode. Encryption key length: 192 bits (48 hexadecimal characters).

AES

Specifies Advanced Encryption Standard encryption. It uses a 128-bit key. Encryption key length: 32 hexadecimal characters.

BLOWFISH

Specifies Blowfish encryption. It uses a 64-bit block and a variable key length. Encryption key length: 32 to 64 even number of hexadecimal characters.

NONE

Specifies no encryption.

Notes:

- CA Workload Automation DE supports the U.S. Government encryption standard FIPS 140-2 and can be configured to run in a FIPS-compliant mode. Your CA Workload Automation DE environment can be considered FIPS-compliant only if all the components use FIPS-compliant algorithms for encryption and decryption. Currently, only AES and DESEDE algorithms are FIPS-certified. If any of your CA Workload Automation DE components use DES or BLOWFISH, your system is not FIPS-compliant.
- If you specify NONE as the encryption method, you must set the security.cryptkey parameter in the agentparm.txt file to no value. For more information about the security.cryptkey parameter, see the *CA Workload Automation Agent for UNIX, Linux, and Windows Implementation Guide*.

user("*user*")

(Tandem and Open-VMS virtual agents only) Specifies the user name for the virtual agent. It is required to add a virtual agent to the Topology.

userids(userid(id("*id*") password("*password*"))...)

(Optional) Specifies a list of users for running jobs. The users are defined on the agent computer.

id("*id*")

Specifies the user ID. This value is case-sensitive.

password("*password*")

Specifies the encrypted password. This value is case-sensitive.

Note: Not all agents support agent users.

Example: Add an Agent to the Topology

The following example adds a new agent named MYAGENT to the Topology:

```
createagent name("MYAGENT") description("Windows-NT/2000") address("HOSTNAME")
port(7520) template("winnt") version("Release 11.3") encryptkey("1234567890ABCDEF")
encryptionmethod("DES") snmpenabled(true) userids(userid(id("user1")
password("1234567890ABCDEF")) userid(id("user2") password("1234567890ABCDEF")))
```

Example: Add an Open-VMS Virtual Agent

The following example adds an Open-VMS virtual agent named TestAgentVMS to the Topology. The Agent007 parent agent connects to the virtual agent using Telnet.

```
createagent name(TestAgentVMS) description("used to simulate VMS jobs")
template(vms) container(Agent007) address(testmachine.mycompany.com) user("user")
userids(userid(id("id") password("password")))
```

More information:

Issue a Command in the CLI Perspective (see page 10)

createagentgroup Command—Create an Agent Group

You can define agent groups to use load balancing or run a job on all agents in a group by issuing the CREATEAGENTGROUP command.

This command has the following format:

```
createagentgroup name("name") type("type") distribution("distribution")
agents(agent(name("name") loadfactor(loadfactor))...)
```

name("*name*")

Defines the agent group name.

Limits: Up to 50 characters including only alphanumeric characters and underscores (_)

type("*type*")

(Optional) Indicates the type of agents being grouped. Options are Unix and Windows.

distribution("*distribution*")

Specifies the criteria the server uses to allocate work to the agents of the group. Options are the following:

CPU

Indicates that the server allocates work to the agents of the group based on the CPU load.

RANDOM

Indicates that the server allocates work to the agents of the group in a random sequence.

ROUNDROBIN

Indicates that the server allocates work to the agents of the group in a round robin.

agents(agent(name("*name*") loadfactor(*loadfactor*))…)

Specifies the list of agents defined in the agent group.

name("*name*")

Specifies the name of the agent.

loadfactor(*loadfactor*)

Indicates the potential load on the agent.

Default: 1

Limits: 1 or higher

Example: Create an Agent Group

The following example creates an agent group named AG3 that contains two Windows agents:

```
createagentgroup name("AG3") type("Windows") distribution("CPU")
agents(agent(name("AGENT1") loadfactor(10)) agent(name("AGENT2") loadfactor(2)))
```

More information:

Issue a Command in the CLI Perspective (see page 10)

deleteagent Command—Remove an Agent from the Topology

You can remove an agent from the Topology if you no longer need to administer the agent by issuing the DELETEAGENT command.

Note: If you remove a parent agent that contains virtual agents, the parent and all its virtual agents are removed.

This command has the following format:

```
deleteagent name("name")
```

name("*name*")

Specifies the name of the agent to be removed from the Topology.

Example: Remove an Agent from the Topology

The following example removes an agent named AGENTRX from the Topology:

```
deleteagent name("AGENTRX")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

deleteagentgroup Command—Delete an Agent Group from the Topology

You can delete an agent group from the Topology if you no longer need to administer the agent group by issuing the DELETEAGENTGROUP command.

Note: If you delete an agent group, the agents belonging to the agent group are not deleted.

This command has the following format:

```
deleteagentgroup name("name")
```

name("*name*")

Specifies the name of the agent group to be deleted.

Example: Delete an Agent Group from the Topology

The following example deletes an agent group named AGENT05 from the Topology:

```
deleteagentgroup name("AGENT05")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

flushagentmsgqueue Command—Clear Agent Receiver Messages

Agent receiver messages are messages sent to an agent to tell it what workload needs to be processed. These messages are queued to wait for processing. You can clear these pending messages by issuing the FLUSHAGENTMSGQUEUE command.

```
flushagentmsgqueue agent("agent") [force(true|false)]
```

agent("*agent*")

Specifies the name of the agent that you want to clear the pending messages for.

force(true|false)

(Optional) Specifies whether to clear the messages if the agent is defined in the Topology. Options are the following:

true

Clears the messages even if the agent is defined in the Topology. Before you clear the pending messages, you must quiesce the agent.

false

Does not clear the messages if the agent is defined in the Topology.

Default: force(false)

Example: Clear Agent Receiver Messages

- The following example clears all of the workload processing messages in the queue for an undefined agent named PSAGENT:

```
flushagentmsgqueue agent("PSAGENT")
```

- The following example clears all of the workload processing messages in the queue for a defined agent named DEFINED_AGENT, assuming the agent is quiesced:

```
flushagentmsgqueue agent("DEFINED_AGENT") force(true)
```

More information:

Issue a Command in the CLI Perspective (see page 10)

listagent Command—Display Status Information for Agents

You can view the agent status information by issuing the LISTAGENT command. You can filter the status information by agent name or by status (active, inactive, quiesced) and view detailed status information.

This command the following format:

```
listagent agent("agent") [status("status")] [type("type")] [verbose(true|false)]
```

agent("*agent*")

Specifies the name of the agent to list.

Default: All agents are listed.

Note: You can include a wildcard for a partial name. For example, AG* displays all the agents whose names start with AG. * represents any string of zero or more characters. ? represents any single character.

status("*status*")

(Optional) Lists agents by status. Options are active, inactive, and quiesced.

type("*type*")

(Optional) Lists agents by type. Options are the following:

- appserv—Application Services
- db—Database
- informatica—Informatica
- microfocus—Micro Focus
- mssqlserver—Microsoft SQL Server
- oracle—Oracle E-Business Suite
- os400— i5/OS
- peoplesoft—PeopleSoft
- remoteexecution—Remote Execution
- sapr3—SAP-R/3
- tandem—Tandem (virtual)
- unix—UNIX
- virtual—Virtual
- vms—Open-VMS (virtual)
- winnt—Windows
- ws—Web Services
- zos—z/OS

verbose(true|false)

(Optional) Indicates whether to display detailed status information. Options are the following:

true

Displays detailed status information

false

Does not display detailed status information

Default: verbose(false)

Examples: Display Status Information for Agents

- The following example displays the status of an agent:

```
listagent agent("agent")
AGENT
Status: Inactive
```

- The following example displays detailed status information for an agent:

```
listagent agent("agent") verbose(true)
AGENT
Status: Inactive
Quiesced: No
Address: EManero1
Port: 7520
Type: Windows-NT/2000
Version: Release 7
Operating System: Windows
Character Code: ASCII
SNMP Enabled: No
Heartbeat Frequency: 2
Strong Encryption Enabled: No
```

- The following example displays detailed status information for all the agents:

```
listagent agent("*") verbose(true)
WFOX24
Status: Active
Quiesced: No
Address: chiki03-test
Port: 8484
Type: Windows
Version: Release 11.3
Operating System: Windows
Character Code: ASCII
SNMP Enabled: No
Heartbeat Frequency: 5
Encryption Method: DES
```

```
WFOX25
Status: Active
Quiesced: No
Address: wade01
Port: 8484
Type: Windows
Version: Release 11.3
Operating System: Windows
Character Code: ASCII
SNMP Enabled: No
Heartbeat Frequency: 5
Encryption Method: DES

SFOX24G
Status: Inactive
Quiesced: No
Address: HOSTNAME
Port: 7520
Type: Windows-NT/2000
Version: Release 11.3
Operating System: Windows
Character Code: ASCII
SNMP Enabled: No
Heartbeat Frequency: 5
Encryption Method: DES
```

More information:

Issue a Command in the CLI Perspective (see page 10)

listagentgroup Command—Display Information for Agent Groups

You can view the information for an agent group or all agent groups by issuing the LISTAGENTGROUP command. You can filter the information by agent group name or by type (UNIX or Windows) and view detailed information.

This command has the following format:

```
listagentgroup agentgroup("agentgroup") [type("type")]
```

agentgroup("*agentgroup*")

Specifies the name of the agent group to list.

Note: You can include a wildcard for a partial name. For example, AG* displays all the agent groups whose names start with AG. * represents any string of zero or more characters. ? represents any single character.

type("*type*")

(Optional) Indicates the type of agents being grouped. Options are Unix and Windows.

Examples: Display Information for Agent Groups

- The following example displays the details of an agent group named AG3:

```
listagentgroup agentgroup("AG3") type("winnt")
Agent group name: AG3
Type: winnt
Distribution: CPU
Agents:
Agent name: AGENT1
Load factor: 10
Agent name: AGENT2
Load factor: 2
```

- The following example displays the details of all the agent groups:

```
listagentgroup agentgroup("*")

Agent group name: AG3
Type: winnt
Distribution: CPU
Agents:
Agent name: AGENT1
Load factor: 10
Agent name: AGENT2
Load factor: 2

Agent group name: AR405
Type: winnt
Distribution: CPU
Agents:
Agent name: KAGENT
Load factor: 1
Agent name: WFOX24
Load factor: 1

Agent group name: UNIX222
Type: unix
Distribution: ROUNDROBIN
Agents:
Agent name: DAGENT
Load factor: 1
Agent name: SFOX24
Load factor: 1
```

More information:

Issue a Command in the CLI Perspective (see page 10)

purgeagentlogs Command—Delete Agent Logs

You can send a message to a specified agent to delete the agent logs. The agent logs are located in the log subdirectory in the directory where the agent is installed. To send a message to delete agent logs, issue the PURGEAGENTLOGS command.

Note: This command only sends a request to the agent. The agent may not delete the logs as requested.

This command has the following format:

```
purgeagentlogs agent("agent")
```

agent("*agent*")

Specifies the name of the agent that you want to delete logs for.

Example: Delete Agent Logs

The following example requests that the agent logs get deleted:

```
purgeagentlogs agent("agent")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

quiesceagent Command—Quiesce an Agent

You can quiesce an agent by issuing the QUIESCEAGENT command to prevent the server from temporarily sending messages or submitting jobs to the agent. When the agent is in Quiesced state, you cannot submit jobs to the agent. However, jobs that are running on the agent when it is quiesced continue to run to completion.

This command has the following format:

```
quiesceagent agent("agent")
```

agent("*agent*")

Specifies the name of the agent that you want to quiesce.

Example: Quiesce an Agent

The following example quiesces the TE_1 agent:

```
quiesceagent agent("TE_1")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

refreshagentsecurity Command—Refresh an Agent Security File

Changes to the security of an agent, such as the addition of a new FTP user, only take effect after the agent security file is refreshed. You can refresh the security file of an agent by issuing the REFRESHAGENTSECURITY command.

This command has the following format:

```
refreshagentsecurity agent("agent")
```

agent("*agent*")

Specifies the name of the agent that you want to refresh the security file for.

Example: Refresh an Agent Security File

The following example refreshes an agent security file:

```
refreshagentsecurity agent("agent")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

setagentproperty Command—Set the Log Level of an Agent

You can change the log level of an agent remotely without modifying the agentparm.txt file and restarting the agent. To change the log level of an agent remotely, issue the SETAGENTPROPERTY command.

This command has the following format:

```
setagentproperty agent("agent") property("log.level=level")
[persistent(true|false)]
```

agent("*agent*")

Specifies the name of the agent.

property("log.level=*level*")

Specifies the new log level. You can specify a log level between 0 and 5.

Example: log.level=5

persistent(true|false)

(Optional) Specifies whether the log level change is permanent. Options are the following:

true

Specifies that the log.level parameter in the agentparm.txt file is updated with the new log level. A backup copy of the original agentparm.txt file is created with the name agentparm.txt.manager.date_time_stamp.txt.

false

Specifies that the log level is changed for the current session only. The log level is reset to the level defined in the agentparm.txt file the next time the agent is restarted.

Default: persistent(false)

Example: Set a Log Level of an Agent

The following example sets the log level of an agent to level 5:

```
setagentproperty agent("agent") property("log.level=5")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

stopagent Command—Stop an Agent

You can stop a specific agent or all agents by issuing the STOPAGENT command.

This command has the following format:

```
stopagent agent("agent")
```

agent("*agent*")

Specifies the name of the agent to stop. You can specify agent("[all]") to stop all agents.

Example: Stop an Agent

The following example stops the TE_1 agent:

```
stopagent agent("TE_1")
```

Example: Stop all Agents

The following example stops all agents:

```
stopagent agent("[all]")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

unquiesceagent Command—Unquiesce an Agent

You can unquiesce an agent by issuing the UNQUIESCEAGENT command. You can unquiesce an agent to release a quiesced agent. The server can then resume sending messages and submitting jobs to the agent.

This command has the following format:

```
unquiesceagent agent("agent")
```

agent("*agent*")

Specifies the name of the agent that you want to unquiesce.

Example: Unquiesce an Agent

The following example quiesces the TE_1 agent:

```
unquiesceagent agent("TE_1")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

updateagent Command—Update an Agent in the Topology

You can update the properties of an agent in the Topology by issuing the UPDATEAGENT command.

This command has the following format:

```
updateagent name("name") [description("description")] [container("container"]
[address("address")] [port(port)] [charcode("charcode")] [version("version")]
[encryptkey("encryptkey")] [serveralias("serveralias")]
[fromagentencryptkey("fromagentencryptkey")] [snmpenabled(true|false)]
[heartbeatfrequency(heartbeatfrequency)] [heartbeatmaxretry(heartbeatmaxretry)]
[persistmgrchange(true|false)] [encryptionmethod("encryptionmethod")]
[user("user")] [userids(userid(id("id") password("password"))...)]
```

name("*name*")

Specifies the name of the agent to be updated.

description("*description*")

(Optional) Defines a description of the agent.

container("*container*")

(Tandem and Open-VMS virtual agents only) Specifies the name of the real (parent) agent that the virtual agent belongs to. It is required to add a virtual agent to the Topology.

address("*address*")

(Optional) Specifies the IP address or DNS name of the computer where the agent is installed.

port(*port*)

(Optional) Specifies the port number that the agent uses to listen for traffic. The port number is not supported on virtual agents.

Default: 7520

Limits: 1024-65534

Note: This port number must match the communication.inputport parameter in the agentparm.txt file.

charcode("*charcode*")

(Optional) Specifies the character set. The character set is not supported on virtual agents. Options are ASCII and EBCDIC.

Default: EBCDIC (z/OS agent); ASCII (all other non-virtual agents)

version("*version*")

(Optional) Identifies the release number of the agent. The release number is not supported on virtual agents.

Note: The value depends on the agent type. You can find the list of values in the Release number drop-down list when you add an agent to the Topology in CA WA Desktop Client.

encryptkey("*encryptkey*")

(Optional) Specifies the encryption key the agent uses to communicate with CA Workload Automation DE. The encryption key is not supported on virtual agents.

Notes:

- CA Workload Automation DE and the agent must have the same encryption key to communicate. The agent's encryption key is stored in a text file (encrypted). The path to that file is set by the security.cryptkey parameter in the agentparm.txt file. If the keys are different, the agent and CA Workload Automation DE cannot communicate and an AGENTDOWN state occurs when you try to run workload.
- If you specify NONE in the encryptionmethod operand, set this value to NOENCRYPTION.

serveralias("*serveralias*")

(z/OS agents only) Specifies the name the agent uses to communicate with the server. It is required to add a z/OS agent to the Topology.

Limits: Up to 16 characters

fromagentencryptkey("*fromagentencryptkey*")

(z/OS agents only) Specifies the encryption key used from agent to server. It is defined for the COMMCHAN initialization parameter in the agent definition data set. It is required to add a z/OS agent to the Topology.

snmpenabled(true|false)

(Optional) Indicates whether SNMP is enabled. This option is not supported on virtual agents.

Default: false

Note: If you specify true, you need to update the SNMP information in the agentparm.txt file.

heartbeatfrequency(*heartbeatfrequency*)

(Optional) Specifies the frequency you want the server to send the heartbeat signal to in minutes. This option is not supported on virtual agents.

Default: 5

Limits: 0 and above

Note: If you want individual agents to have their own heartbeat frequencies, you can set the internal shared parameter named Global agent heartbeat interval in minutes to 0 (zero).

heartbeatmaxretry(*heartbeatmaxretry*)

(Optional) Specifies the number of heartbeat signals the server attempts before it sends an SNMP message indicating agent inactivity. This option is not supported on virtual agents.

Default: 1

Limits: 1 and above

persistmgrchange(true|false)

(Optional) Indicates whether the changes to server properties that affect agent communication are permanently changed on the agent. If you change the ID, address, or port of the server in the Topology, the server sends the agent the updated server connection information at the next heartbeat signal, allowing the server and agent to communicate. Options are as follows:

true

Updates the communication parameters in the agentparm.txt file with changes made to the corresponding properties in the server Topology. For example, if you change the server address and port in the Topology, the communication.manageraddress_*n* and communication.managerport_*n* parameters in the agentparm.txt file are updated with the new server address and port.

false

Updates the server connection information every time the agent is restarted, allowing the server and agent to communicate for that session only. The communication changes are not saved in the agentparm.txt file.

Note: This option only applies to Release 7 agents and higher and is not supported on virtual agents.

encryptionmethod("*encryptionmethod*")

(Optional) Specifies the encryption method the CA Workload Automation DE uses to encrypt messages. The encryption method is not supported on virtual agents. Options are the following:

DES

Specifies Data Encryption Standard encryption. It uses a 56-bit key. Encryption key length: 56 bits (16 hexadecimal characters).

DESEDE

Specifies 3DES encryption. It uses the DES algorithm in EDE (encrypt-decrypt-encrypt) mode. Encryption key length: 192 bits (48 hexadecimal characters).

AES

Specifies Advanced Encryption Standard encryption. It uses a 128-bit key. Encryption key length: 32 hexadecimal characters.

BLOWFISH

Specifies Blowfish encryption. It uses a 64-bit block and a variable key length. Encryption key length: 32 to 64 even number of hexadecimal characters.

NONE

Specifies no encryption.

Notes:

- CA Workload Automation DE supports the U.S. Government encryption standard FIPS 140-2 and can be configured to run in a FIPS-compliant mode. Your CA Workload Automation DE environment can be considered FIPS-compliant only if all the components use FIPS-compliant algorithms for encryption and decryption. Currently, only AES and DESEDE algorithms are FIPS-certified. If any of your CA Workload Automation DE components use DES or BLOWFISH, your system is not FIPS-compliant.
- If you specify NONE as the encryption method, you must set the security.cryptkey parameter in the agentparm.txt file to no value. For more information about the security.cryptkey parameter, see the *CA Workload Automation Agent for UNIX, Linux, and Windows Implementation Guide*.

user("*user*")

(Tandem and Open-VMS virtual agents only) Specifies the user name for the virtual agent. It is required to add a virtual agent to the Topology.

userids(*userid*(id("*id*") password("*password*"))...)

(Optional) Specifies a list of users for running jobs. The users are defined on the agent computer.

id("*id*")

Specifies the user ID. This value is case-sensitive.

password("*password*")

Specifies the encrypted password. This value is case-sensitive.

Note: Not all agents support agent users.

Example: Update an Agent in the Topology

The following example updates an agent named MYAGENT in the Topology:

```
updateagent name("MYAGENT") description("new agent description")
userids(userid(id("existing_user1") password("1234567890ABCDEF"))
userid(id("existing_user2") password("1234567890ABCDEF")))
```

Example: Update a Virtual Agent in the Topology

The following example updates the parent agent and user name of a virtual agent named TestAgentVMS:

```
updateagent name(TestAgentVMS) container(Agent008) user("user1")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

updateagentgroup Command—Update an Agent Group

You can update the properties of an agent group by issuing the UPDATEAGENTGROUP command.

This command has the following format:

```
updateagentgroup name("name") distribution("distribution")
agents(agent(name("name") loadfactor(loadfactor))...)
```

name("*name*")

Specifies the name of the agent group to be updated.

distribution("*distribution*")

Specifies the criteria the server uses to allocate work to the agents of the group. Options are the following:

CPU

Indicates that the server allocates work to the agents of the group based on the CPU load.

RANDOM

Indicates that the server allocates work to the agents of the group in a random sequence.

ROUNDROBIN

Indicates that the server allocates work to the agents of the group in a round robin.

agents(agent(name("*name*") loadfactor(*loadfactor*))…)

Specifies the list of agents defined in the agent group.

name("*name*")

Specifies the name of the agent.

loadfactor(*loadfactor*)

Indicates the potential load on the agent.

Default: 1

Limits: 1 or higher

Example: Update an Agent Group

The following example updates the agent group named AGENTGROUP:

```
updateagentgroup name("AGENTGROUP") distribution("CPU")
agents(agent(name("AGENT1") loadfactor(10)) agent(name("AGENT2") loadfactor(2)))
```

More information:

Issue a Command in the CLI Perspective (see page 10)

Chapter 4: Work with Server Log Files

This section contains the following topics:

applylogprofile Command—Apply a Logging Profile

You can apply a logging profile to the current logging configuration by issuing the APPLYLOGPROFILE command.

This command has the following format:

```
applylogprofile name("name")
```

name("*name*")

Specifies the name of the logging profile that you want to apply to the current logging configuration.

Example: Apply a Logging Profile

The following example applies the logging profile named DEFAULT to the current logging configuration:

```
applylogprofile name("DEFAULT")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

exportauditlog Command—Create an Audit Log Report

The audit log report contains every user-initiated server command. You can create an audit log report by issuing the EXPORTAUDITLOG command. This command creates an audit log in CSV format and a report in HTML format. For each command that a user issues, the audit log records the following information:

- Date—The date on which the user issued the command
- Time—The time at which the user issued the command
- Correlation identifier—The identifier that matches a command with its corresponding response
- Record type—The type of command
- User ID—The User ID that issued the command
- Action type—The type of action that the command represents
- Action friendly name—The name for the command
- Data—The command or response data

This command has the following format:

```
exportauditlog path("path") name("name") [startdate("startdate")]
[endate("enddate")]
```

path("*path*")

Identifies the directory in which the audit report is created.

name("*name*")

Identifies the name of the created audit report.

startdate("*startdate*")

(Optional) Identifies the start date or first date in the report in the format *yyyymmdd*.

endate("*enddate*")

(Optional) Identifies the end date or last date in the report in the format *yyyymmdd*.

Note: If you enter a start date and do not enter an end date, the server exports audit log records between the start date and the end of the audit log.

Example: Create an Audit Log Report

The following example creates an audit log report in the d:\wutemp directory starting from August 1, 2007:

```
exportauditlog path("d:\\wutemp") name("auditlog") startdate("20070801")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

getlogprofile Command—Display a Logging Profile

You can display the logger identifiers and the corresponding severity thresholds of a logging profile by issuing the GETLOGPROFILE command.

This command has the following format:

```
getlogprofile name("name")
```

name("*name*")

Specifies the name of the logging profile that you want to view the details of.

Example: Display a Logging Profile

The following example displays the details of a logging profile named ESSENTIAL:

```
getlogprofile name("ESSENTIAL")
Profile name: ESSENTIAL
Loggers:
acl:utils - WARN
aet - WARN
afm:removal - WARN
afm:routing - WARN
agent:afm - INFO
alert:manager - WARN
communications - WARN
datacapture - WARN
dm - WARN
essential - INFO
externals - WARN
facade:request - WARN
noise - WARN
outgoing:afms - WARN
parser - WARN
process:flow - WARN
publishsubscribe - WARN
pubsub - WARN
variable:dependency - WARN
wob:management - WARN
ws - WARN
ws:clientsession - WARN
ws:clientsession:subscription - OFF
ws:connections - WARN
```

More information:

Issue a Command in the CLI Perspective (see page 10)

getlogthreshold Command—Display the Severity Threshold of the Logger Identifiers

After you have set the severity threshold for each message category (logger identifier), you can display the severity threshold of the logger identifiers by issuing the GETLOGTHRESHOLD command.

This command has the following format:

```
getlogthreshold [logger("logger")]
```

logger("*logger*")

(Optional) Specifies the logger identifier that you want to display the severity threshold for.

Note: If you do not specify the logger identifier, the severity thresholds of all logger identifiers are displayed.

Examples: Display the Severity Threshold of the Logger Identifiers

- The following example displays the severity threshold of the logger identifier named aet:

```
getlogthreshold logger("aet")
Current severity threshold for logger aet is DEBUG.
```

- The following example displays the severity threshold of all the logger identifiers:

```
getlogthreshold
Loggers and their corresponding severity thresholds:
acl:utils - WARN
aet - WARN
aetl - threshold is not set
afm:removal - INFO
afm:routing - INFO
agent:afm - INFO
alert:manager - INFO
ams:lock:manager - WARN
appl:folder - WARN
appl:lump - INFO
attach:spoolfile - INFO
buffer - ALL
cache:info - WARN
communications - INFO
datacapture - INFO
dm - INFO
dm:actioncommand - INFO
dm:applstate - OFF
dm:commandmanager - WARN
dm:communications - WARN
dm:communications:exception - WARN
dm:controlobject - INFO
dm:critical:path - WARN
dm:emailclient - threshold is not set
dm:publishsubscribe - INFO
```

More information:

Issue a Command in the CLI Perspective (see page 10)

loginfo Command—Display Log Information

You can view log information by issuing the LOGINFO command.

This command has the following format:

```
loginfo log("log")
```

log("*log*")

Specifies the log to display. Options are auditlog and tracelog.

Examples: Display Log Information

- The following example displays information about the trace log:

```
loginfo log("tracelog")
Trace log[available=true, current=D:/Program Files/dSeries
/LogFiles/tracelog.txt, filterids=(0), filelist=(D:/Program
Files/dSeries/LogFiles/tracelog.txt,)]
```

- This following example displays information about the audit log:

```
loginfo log("auditlog")
Auditlog [available: true; current log file:
C:\CAWADE\logs\auditlog.20100428.bin; file=C:\CAWADE\logs\auditlog.bin]
```

More information:

Issue a Command in the CLI Perspective (see page 10)

purgelog Command—Clear the Trace Log File

The server maintains the trace log for storing diagnostic information. To maintain performance, you must clear the trace log files regularly. You can issue the PURGELOG command to reclaim the disk space quickly.

Important! Do not use the PURGELOG command unless there is an emergency situation. Usually, the system administrator gets enough warnings to avert an emergency situation.

This command has the following format:

`purgelog age(`*age*`)`

age(*age*)

Clears trace logs older than the specified number of days.

Limits: 1 and above

Example: Clear the Trace Log File

The following example clears the trace log files from the server that are older than three days:

```
purgelog age(3)
```

More information:

Issue a Command in the CLI Perspective (see page 10)

setlogthreshold Command—Set the Severity Threshold of a Logger Identifier

To identify the types of messages that are logged in the trace log, you can set the severity threshold for each message category (logger identifier). The messages associated with the category with severity lower than the threshold are discarded. For example, if you specify ALL as the severity threshold for a logger identifier, all associated messages are stored in the log file. If you specify INFO as the severity threshold for a logger identifier, only the associated messages with severity INFO and higher are stored in the log file. You can set the severity threshold of a logger identifier by issuing the SETLOGTHRESHOLD command.

This command has the following format:

```
setlogthreshold logger("logger") threshold("threshold")
```

logger("*logger*")

Specifies the logger identifier that you want to set the severity threshold for.

threshold("*threshold*")

Specifies the severity threshold for the logger identifier. The options are as follows:

ALL

Indicates that all the messages are logged in the trace log.

DEBUG

Indicates that debugging, information, warning, error, and fatal messages are logged in the trace log.

INFO

Indicates that information, warning, error, and fatal messages are logged in the trace log.

WARN

Indicates that warning, error, and fatal messages are logged in the trace log.

ERROR

Indicates that error and fatal messages are logged in the trace log.

FATAL

Indicates that only the fatal messages are logged in the trace log.

OFF

Indicates that no messages are logged in the trace log.

Example: Set the Severity Threshold of a Logger Identifier

The following example sets the severity threshold for the logger identifier named essential to DEBUG:

```
setlogthreshold logger("essential") threshold("DEBUG")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

spinlog Command—Archive the Active Trace Log File

You can archive the active trace log file by issuing the SPINLOG command.

This command has the following format:

```
spinlog
```

Note: For more information about archiving the active trace log file, see the *Admin Perspective Help*.

More information:

Issue a Command in the CLI Perspective (see page 10)

Chapter 5: Work with Applications and Jobs

This section contains the following topics:

lastrun Command—Display the Last Run Information of a Job

You can display information about the last runs of a job by issuing the LASTRUN command.

This command has the following format:

```
lastrun job("job") [application("application")] [runs(runs)]
```

job("*job*")

Specifies the job to list.

application("*application*")

(Optional) Specifies the name of the Application containing the job.

runs(*runs*)

(Optional) Defines the number of runs to list.

Default: Previous ten runs of the job.

Example: Display the Last Run Information of a Job

The following example displays the last run information of a job:

```
lastrun job("WINNT1") application("VERIFY")
WINNT1
  Job type:        NT
  Start time:      Fri Oct 26 17:58:54 GMT+05:30 2007
  End time:        Fri Oct 26 17:58:54 GMT+05:30 2007
  Application:     VERIFY.2
  State:           COMPLETE
  Completion Code: 0
WINNT1
  Job type:        NT
  Start time:      Fri Oct 26 17:52:31 GMT+05:30 2007
  End time:        Fri Oct 26 17:52:31 GMT+05:30 2007
  Application:     VERIFY.1
  State:           COMPLETE
  Completion Code: 0
```

More information:

Issue a Command in the CLI Perspective (see page 10)

listaetdata Command—List Average Execution Time Data

You can list average execution time (AET) data by issuing the LISTAETDATA command.

This command has the following format:

```
listaetdata [applicationname("applicationname")] [jobname("jobname")]
[profile("profile")] [olderthan("olderthan")] [showdetail(true|false)]
```

applicationname("*applicationname*")

(Optional) Lists AET data associated with the specified Application name. Specify a regular expression for the Application name.

Example: applicationname("payroll")

Note: To compose a regular expression, follow the rules for Java class java.util.regex.Pattern. You can find these rules on the Internet by searching for "java pattern".

jobname("*jobname*")

(Optional) Lists AET data associated with the specified job name. Specify a regular expression for the job name.

Example: jobname("job1.qual")

Notes:

- If the job has a qualifier, specify *jobname.qualifier*, where *jobname* is the job name and *qualifier* is the job qualifier, for example, JBC.qual2.
- To compose a regular expression, follow the rules for Java class java.util.regex.Pattern. You can find these rules on the Internet by searching for "java pattern".

profile("*profile"*)

(Optional) Lists AET data associated with the specified profile name. Specify a regular expression for the profile name.

Example: profile("last")

Notes:

- To compose a regular expression, follow the rules for Java class java.util.regex.Pattern. You can find these rules on the Internet by searching for "java pattern".
- To list AET data from the default profile, specify a single space.

olderthan("*olderthan"*)

(Optional) Lists the AET data older than a scheduled time.

Example: olderthan("now less 1 month")

Notes:

- If a job profile was updated after the specified time, the AET data for the profile is not returned. For example, assume that a job has two profiles named prof1 and prof 2; prof1 was last updated more than one month ago and prof2 was last updated one week ago. In this example, listaetdata olderthan("now less 1 month") returns AET data for prof1, but not prof2.
- The value must be a schedule criteria statement that resolves to a date and time. For more information about schedule criteria, see the *Define Perspective Help*.

showdetail(true|false)

(Optional) Indicates whether you want to display detailed information on the last ten executions of the job. Options are the following:

true

Displays detailed information on the last ten job executions, including generation number and execution time.

false

Does not display detailed information on the last ten job executions.

Default: showdetail(false)

Examples: List AET Data

- The following example lists detailed AET data for the default job profile:

```
listaetdata profile(' ') showdetail(true)

Application name:[EXT12] job name:[EXT1] profile name:[ ]
agent[ESP_CYBER-TEST03_CA_COM_7500] last update:[2010-06-11 17:45:12.0] average
execute time: [8 seconds] average life time: [3 seconds]
details:
    Generation #2:8 seconds
    Generation #3:10 seconds
    Generation #4:9 seconds
    Generation #5:7 seconds
    Generation #6:9 seconds
    Generation #7:10 seconds
Application name:[EXT5] job name:[EXT5] profile name:[ ] agent[MB9_AGT] last
update:[2010-06-11 17:45:09.0] average execute time: [4 seconds] average life
time: [6 seconds]
details:
    Generation #2:5 seconds
    Generation #3:5 seconds
    Generation #4:5 seconds
    Generation #5:5 seconds
    Generation #6:4 seconds
    Generation #7:5 seconds
```

- The following example lists the AET data of all the jobs in the VERIFY Application that have 1 as the last character in the job name:

```
listaetdata applicationname("VERIFY") jobname(".+1")

Application name:[VERIFY] job name:[LINUX1] profile name:[ ] agent[HP_AG] last
update:[2010-06-07 01:25:26.273] average execute time: [3 seconds] average life
time: [4 seconds]
```

- The following example lists the AET data that is older than a month:

```
listaetdata olderthan("now less 1 month")
Application name:[TEST] job name:[LINK0] profile name:[PROF TUE]
agent[ESP_CYBER-TEST03_CA_COM_7500] last update:[2010-04-28 10:21:05.0] average
execute time: [5 seconds] average life time: [3 seconds]
Application name:[CYBER] job name:[WIN] profile name:[PROF TUE]
agent[ESP_CYBER-TEST03_CA_COM_7500] last update:[2010-04-27 14:21:05.0] average
execute time: [4 seconds] average life time: [2 seconds]
```

- The following example lists the AET data for the LINK0 job in the TAG Application:

```
listaetdata applicationname("TAG") jobname("LINK0")
Application name:[TAG] job name:[LINK0] profile name:[MONDAY]
agent[ESP_CYBER-TEST03_CA_COM_7500] last update:[2010-05-28 10:21:05.0] average
execute time: [5 seconds] average life time: [2 seconds]
```

- The following example lists the AET data of all the jobs in the RSV Application that have J as the first character and C as the last character in the job name:

```
listaetdata applicationname("RSV") jobname("J.+C")

Application name:[RSV] job name:[JC] profile name:[ ] agent[HP_AG] last
update:[2010-06-07 01:25:26.273] average execute time: [3 seconds] average life
time: [4 seconds]

Application name:[RSV] job name:[JBBC] profile name:[ ] agent[HP_AG] last
update:[2010-06-07 01:25:26.273] average execute time: [5 seconds] average life
time: [2 seconds]
```

- The following example lists the AET data of all the jobs whose names start with abc, follow with a single character, and end with def:

```
listaetdata jobname("abc.def")

Application name:[TTT] job name:[abc.def] profile name:[ ] agent[MS4] last
update:[2010-06-09 09:25:26.273] average execute time: [2 seconds] average life
time: [4 seconds]

Application name:[TTT] job name:[abcxdef] profile name:[ ] agent[HP_AG] last
update:[2010-06-09 06:25:26.273] average execute time: [8 seconds] average life
time: [2 seconds]
```

More information:

Issue a Command in the CLI Perspective (see page 10)

listapplication Command—Display Status Information for an Application

You can view the status information for an Application including state, anticipated end time, and percentage completion by issuing the LISTAPPLICATION command.

This command has the following format:

```
listapplication application("application") [summary(true|false)] [gen(gen)]
[state("state")] [oldest(true|false)] [job(joblist("joblist"))]
[subappl("subappl")] [critpath(true|false)]
```

application("*application*")

Specifies the name of the Application to list.

summary(true|false)

(Optional) Specifies whether to display the Application summary. Options are the following:

true

Displays the application summary.

false

Does not display the application summary.

Default: summary(false)

gen(*gen*)

(Optional) Identifies the generation of the Application. You can specify 0 to list the most recent generation. You can specify a negative generation number to display a previous generation relative to the current generation.

Default: All active Applications are displayed.

Example: -1 displays the previous generation.

state("*state*")

(Optional) Specifies whether to list active, complete, or all Application generations. Options are the following:

active

Lists active Application generations.

complete

Lists complete Application generations.

all

Lists all Applications generations.

Default: All active generations of the specified Application are listed.

oldest(true|false)

(Optional) Specifies whether to list only the oldest incomplete generation. Options are the following:

true

Lists only the oldest incomplete generation.

false

Lists all incomplete generations.

Default: oldest(false)

job(joblist(*"joblist"*))

(Optional) Lists a specific job in the Application. You can specify multiple jobs separated by commas.

Example: job(joblist("job0,job1,job2"))

subappl("*subappl*")

(Optional) Limits the listing to the specified subApplication.

critpath(true|false)

(Optional) Specifies whether to list only jobs on the critical path. This parameter only works with active Applications. Options are the following:

true

Lists only jobs on the critical path.

false

Lists all jobs.

Default: critpath(false)

Example: Display the Status Information of an Application

The following example displays the Application summary of generation 1 of the Application app_name:

```
listapplication application("app_name") summary(true) gen(1)
```

More information:

Issue a Command in the CLI Perspective (see page 10)

purgeaetdata Command—Purge Average Execution Time Data

You can delete average execution time (AET) data by issuing the PURGEAETDATA command. All the information on the job execution is removed from the database.

This command has the following format:

```
purgeaetdata [applicationname("applicationname")] [jobname("jobname")]
[profile("profile")] [olderthan("olderthan")]
```

applicationname("*applicationname*")

(Optional) Deletes AET data associated with the specified Application name. Specify a regular expression for the Application name.

Example: applicationname("payroll")

Note: To compose a regular expression, follow the rules for Java class java.util.regex.Pattern. You can find these rules on the Internet by searching for "java pattern".

jobname("*jobname*")

(Optional) Deletes AET data associated with the specified job name. Specify a regular expression for the job name.

Example: jobname("job1.qual")

Notes:

- If the job has a qualifier, specify *jobname.qualifier*, where *jobname* is the job name and *qualifier* is the job qualifier, for example, JBC.qual2.
- To compose a regular expression, follow the rules for Java class java.util.regex.Pattern. You can find these rules on the Internet by searching for "java pattern".

profile("*profile*")

(Optional) Deletes AET data associated with the specified profile name. Specify a regular expression for the profile name.

Example: profile("last")

Notes:

- To compose a regular expression, follow the rules for Java class java.util.regex.Pattern. You can find these rules on the Internet by searching for "java pattern".
- To delete AET data from the default profile, specify a single space.

olderthan("*olderthan*")

(Optional) Deletes AET data older than a scheduled time.

Example: olderthan("now less 1 month")

Notes:

- If a job profile was updated after the specified time, the AET data for the profile is not deleted. For example, assume that a job has two profiles named prof1 and prof 2; prof1 was last updated more than one month ago and prof2 was last updated one week ago. In this example, purgeaetdata olderthan("now less 1 month") deletes AET data for prof1, but not prof2.
- The value must be a schedule criteria statement that resolves to a date and time. For more information about schedule criteria, see the *Define Perspective Help*.

Examples: Purge AET Data

- The following example deletes the AET data of all the jobs in the VERIFY Application that have 1 as the last character in the job name:

  ```
  purgeaetdata applicationname("verify") jobname(".+1")
  ```

- The following example deletes the AET data for the default job profile:

  ```
  purgeaetdata profile(' ')
  ```

- The following example deletes the AET data that is older than a month:

  ```
  purgeaetdata olderthan("now less 1 month")
  ```

More information:

Issue a Command in the CLI Perspective (see page 10)

Chapter 6: Work with Events

This section contains the following topics:

bypassevent Command—Bypass a Future Execution of an Event

You can bypass a future scheduled execution of an Event to prevent the server from triggering it at its scheduled time. You can bypass an Event execution by issuing the BYPASSEVENT command.

This command has the following format:

```
bypassevent event("eventprefix.eventname") [at("yyyy.MMM.dd HH:mm:ss:SSS")]
[triggeradd("triggeradd")]
```

event("*eventprefix.eventname*")

Identifies the Event you want to bypass.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

at("*yyyy.MMM.dd HH:mm:ss:SSS*")

(Optional) Specifies the scheduled time of the Event execution to bypass.

Note: You can issue the LISTEVENTSCHEDULE command to view the scheduled time of the Event.

triggeradd("*triggeradd*")

(Optional) Specifies an Event execution that was manually triggered as an addition to the schedule (trigger add) to bypass.

Note: You can issue the LISTEVENTSCHEDULE command to view the Event execution.

Note: Specify the at operand or the triggeradd operand, but not both. If you do not specify either operand, the server bypasses the next execution of the Event.

Example: Bypass the Next Scheduled Execution of an Event

Suppose that an Event triggers at 7 p.m. daily. At 3 p.m., you decide that the Event should not run that night. To prevent the Event from running, you bypass it using the following command:

```
bypassevent event("it.applsmall")
```

Example: Bypass a Future Scheduled Execution of an Event

This example bypasses the cybermation.verify Event execution scheduled for November 17, 2010 at 9:00 p.m.:

```
bypassevent event(cybermation.verify) at('2010.Nov.17 21:00:00.000')
```

More information:

Issue a Command in the CLI Perspective (see page 10)
listeventschedule Command—List Next Scheduled Events (see page 83)

hold Command—Hold an Event

You can hold an Event if you want to prevent the Event from triggering. The Event remains in a held state until you release it. You can hold an Event by issuing the HOLD command.

This command has the following format:

```
hold event("eventprefix.eventname")
```

event("*eventprefix.eventname*")

Identifies the Event you want to hold.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

Example: Hold an Event

The following example holds an Event:

```
hold event("cybermation.verify")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

listevent Command—List Events

You can update, control, or check the status of Events defined on your system. You can list Events by issuing the LISTEVENT command.

This command has the following format:

```
listevent [name("eventprefix.eventname")]
```

name("*eventprefix.eventname*")

(Optional) Identifies the Event you want to list.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

Default: All Events

Note: You can include a wildcard for a partial name. For example, EV* displays all the Events with prefixes that start with EV. * represents any string of zero or more characters. ? represents any single character.

Example: Display an Event

The following example displays an Event:

```
listevent name("scheduser.app002")
Prefix and name: SCHEDUSER.APP002
Application name: APP002
Type: Date-Time/Manual
Next scheduled time: Tue Sep 04 15:00:00 EDT 2007
Hold count: 0
Suspend count: 0
Status:
Last modified time: Mon Sep 03 14:41:05 EDT 2007
Last modified by: SCHEDUSER
```

More information:

Issue a Command in the CLI Perspective (see page 10)

listeventschedule Command—List Next Scheduled Events

You can display a list of Events scheduled within a specific time period (the default is the next 24 hours). For each scheduled Event execution, the Event name, its scheduled time, and its Application parameters are displayed. Additional details are displayed for bypassed and manually triggered Event executions. You can display scheduled Events by issuing the LISTEVENTSCHEDULE command.

This command has the following format:

```
listeventschedule event("eventprefix.eventname") from("from") to("to") count(count)
```

event("*eventprefix.eventname*")

Identifies the Event you want to list scheduled executions for.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

Note: You can include the asterisk (*) wildcard for a partial name. For example, EV* displays all the Events with prefixes that start with EV. The asterisk represents any string of zero or more characters.

from("*from*")

Specifies the starting time and date of the schedule's period.

to("*to*")

Specifies the ending time and date of the schedule's period.

count(*count*)

Specifies the number of scheduled Event executions to list.

Limits: 1 and greater

Examples: List Next Scheduled Events

- The following example lists the next three scheduled test.test Event executions between two dates:

  ```
  listeventschedule event("test.test") from("2010/10/11") to("2010/10/13")
  count(3)
  ```

- The following example lists the next scheduled test.test Event execution in the next three hours:

  ```
  listeventschedule event("test.test") from("now") to("now plus 3 hours") count(1)
  ```

More information:

Issue a Command in the CLI Perspective (see page 10)

release Command—Release a Held Event

Before a held Event can be triggered, you must release it. If the Event missed a scheduled execution while on hold, the server immediately triggers the Event once, regardless of the number of missed executions. You can release an Event by issuing the RELEASE command.

This command has the following format:

```
release event("eventprefix.eventname")
```

event("*eventprefix.eventname*")

Identifies the Event you want to release.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

Example: Hold and Release an Event

Consider the following scenario:

- At 3 p.m., you are asked to postpone an Event named ACCNT.PAYROLL. ACCNT.PAYROLL is processed at 4 p.m. daily.
- You hold the Event named ACCNT.PAYROLL.
- At 4 p.m., the server sees that the Event is on hold and does not process ACCNT.PAYROLL.
- At 5 p.m., you are instructed to let ACCNT.PAYROLL process.
- You release the Event named ACCNT.PAYROLL.

By default, the Event then processes because the server knows the Event missed its scheduled execution at 4 p.m.

The following example releases an Event:

```
release event("ACCNT.PAYROLL")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

resume Command—Resume a Suspended Event

When you want a suspended Event to resume its execution, you must resume it. You can resume a suspended Event by issuing the RESUME command.

Note: An Event can be automatically suspended and resumed at certain times. For more information about automatically suspending and resuming an Event, see the *Define Perspective Help*.

This command has the following format:

```
resume event("eventprefix.eventname")
```

event("*eventprefix.eventname*")

Identifies the Event you want to resume.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

Example: Resume a Suspended Event

The following example resumes a suspended Event:

```
resume event("cybermation.verify")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

scheduleallevents Command—Reschedule All Events Containing a Schedule Statement

You can reschedule all Events containing a schedule statement by issuing the SCHEDULEALLEVENTS command.

This command has the following format:

```
scheduleallevents [verbose(true|false)]
```

verbose

(Optional) Specifies whether to display details. Options are the following:

true

Displays details of rescheduled Events.

false

Does not display details of rescheduled Events.

Default: verbose(false)

Example: Reschedule all Events Containing a Schedule Statement

The following example reschedules all Events containing a schedule statement:

```
scheduleallevents
```

More information:

Issue a Command in the CLI Perspective (see page 10)

suspend Command—Suspend an Event

To prevent an Event from triggering, you can instruct the server to suspend it. The Event remains suspended until you resume it.

The server handles suspended Events as follows:

- For scheduled Events, the server does not schedule the Event until the Event is resumed. The server ignores any missed scheduled executions while the Event is suspended.
- For monitor Events, the server does not trigger the Event until the Event is resumed. The agent monitoring for the Event stops monitoring while the Event is suspended.
- For manual Events, you cannot trigger the Event until the Event is resumed.

Note: An Event can be automatically suspended and resumed at certain times. For more information about automatically suspending and resuming an Event, see the *Define Perspective Help*.

This command has the following format:

```
suspend event("eventprefix.eventname")
```

event("*eventprefix.eventname*")

Identifies the Event you want to suspend.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

Example: Suspend an Event

The following example suspends an Event:

```
suspend event("cybermation.verify")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

triggeradd Command—Trigger an Event as an Addition to the Schedule

You can manually trigger an Event as an addition to the schedule by issuing the TRIGGERADD command.

Note: For more information about how Events can be triggered, see the *Services Perspective Help*.

This command has the following format:

```
triggeradd event("eventprefix.eventname") [at("at")] [hold(true|false)] [u1("u1")]
[u2("u2")] [u3("u3")] [u4("u4")] [rootjobs(job("job"))] [vars(prop(name("name")
val("val")),...)+]
```

event("*eventprefix.eventname*")

Identifies the Event you want to trigger as an addition to the schedule.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

at("*at*")

(Optional) Defines the trigger time.

Default: now

hold(true|false)

(Optional) Specifies whether to trigger the Event on hold. Options are the following:

true

Triggers the Event on hold.

false

Does not trigger the Event on hold.

Default: hold(false)

u1("*u1*")

(Optional) Defines the 1st user parameter.

u2("*u2*")

(Optional) Defines the 2nd user parameter.

u3("*u3*")

(Optional) Defines the 3rd user parameter.

u4("*u4*")

(Optional) Defines the 4th user parameter.

rootjobs(job("*job*"))

(Optional) Specifies a root job in the Application. You can specify multiple root jobs separated by commas.

Example: job("winnt3,winnt2")

vars(prop(name("name") val("val")),...)

(Optional) Identifies a list of property/value pairs to pass to the Event. Separate each property/value pair with a comma.

Example: prop(name("p1") val("aa")), prop(name("p2") val("bb"))

Example: Trigger an Event as an Addition to the Schedule

The following example lets you trigger an Event in addition to the schedule:

```
triggeradd event("cybermation.verify")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

triggerreplace Command—Trigger an Event to Replace the Next Execution

When you trigger an Event to run an Application at a specified time, you can replace the Event's next scheduled or expected execution by issuing the TRIGGERREPLACE command.

This command has the following format:

```
triggerreplace event("eventprefix.eventname") [at("at")] [hold(true|false)]
[u1("u1")] [u2("u2")] [u3("u3")] [u4("u4")] [rootjobs(job("job"))]
```

event("*eventprefix.eventname*")

Identifies the Event you want to trigger as a replacement of the Event's next scheduled or expected execution.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

at("*at*")

(Optional) Defines the trigger time.

Default: now

hold(true|false)

(Optional) Specifies whether to trigger the Event on hold. Options are the following:

true

Triggers the Event on hold.

false

Does not trigger the Event on hold.

Default: hold(false)

u1("*u1*")

(Optional) Defines the 1st user parameter.

u2("*u2*")

(Optional) Defines the 2nd user parameter.

u3("*u3*")

(Optional) Defines the 3rd user parameter.

u4("*u4*")

(Optional) Defines the 4th user parameter.

rootjobs(job("job"))

(Optional) Specifies a root job in the Application. You can specify multiple root jobs separated by commas.

Example: job("winnt3,winnt2")

Example: Trigger an Event to Replace the Next Execution

The following example lets you trigger an Event to replace the next execution:

```
triggerreplace event("cybermation.verify")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

unbypassevent Command—Unbypass a Future Execution of an Event

You can cancel the bypass of a future scheduled execution of an Event if it has been bypassed. The server triggers the Event at its scheduled time. You can unbypass an Event execution by issuing the UNBYPASSEVENT command.

This command has the following format:

```
unbypassevent event("eventprefix.eventname") [at("yyyy.MMM.dd HH:mm:ss:SSS")]
[triggeradd("triggeradd")]
```

event("*eventprefix.eventname*")

Identifies the Event you want to unbypass.

eventprefix

Specifies the Event prefix.

eventname

Specifies the Event name.

at("*yyyy.MMM.dd HH:mm:ss:SSS*")

(Optional) Specifies the scheduled time of the Event execution to unbypass.

Note: You can issue the LISTEVENTSCHEDULE command to view the scheduled time of the Event.

triggeradd("*triggeradd*")

(Optional) Specifies an Event execution that was manually triggered as an addition to the schedule (trigger add) to unbypass.

Note: You can issue the LISTEVENTSCHEDULE command to view the Event execution.

Note: Specify the at operand or the triggeradd operand, but not both. If you do not specify either operand, the server unbypasses the next execution of the Event.

Examples: Unbypass a Future Scheduled Execution of an Event

- The following example unbypasses the next scheduled execution of the it.applsmall Event:

  ```
  unbypassevent event("it.applsmall")
  ```

- The following example unbypasses the Event execution scheduled to be bypassed at November 17, 2010 at 9:00 p.m.:

  ```
  unbypassevent event(cybermation.verify) at('2010.Nov.17 21:00:00.000')
  ```

More information:

Issue a Command in the CLI Perspective (see page 10)
listeventschedule Command—List Next Scheduled Events (see page 83)

Chapter 7: Work with Global Variables and Variable Dependencies

This section contains the following topics:

decrementvar Command—Decrement an Integer-valued Global Variable

You can decrement a global variable value by 1 if the value is an integer. To decrement a global variable value, issue the DECREMENTVAR command.

Note: If the specified global variable does not exist, the variable is created and its value is set to -1. If the specified global variable is already assigned the minimum integer value (-2^{63}), its value is set to -1. If the value of the specified global variable is not an integer, you get an error.

The command has the following format:

```
decrementvar name("name") [context("context")]
```

name("*name*")

Defines the name of the global variable. This name must be unique in its variable context. This name is not case sensitive.

Limits: 1-128 alphanumeric or underscore characters. The first character cannot be a number.

context("*context*")

(Optional) Defines the name of the context that the global variable belongs to.

Limits: 1-128 alphanumeric or underscore characters

Default: DEFAULT context

Examples: "payroll", “01_group”

Example: Decrement an Integer-valued Global Variable

Suppose that a global variable named maxnum in the DEFAULT context contains a value of 10. The following example decrements the value of the maxnum variable by 1:

```
decrementvar name("maxnum")
```

```
Command successful.
Value=9
```

More information:

Issue a Command in the CLI Perspective (see page 10)

deletevar Command—Delete a Global Variable

You can delete a global variable if the variable is no longer in use. To delete a global variable, issue the DELETEVAR command.

Note: If you delete a global variable that is specified in a %VAR statement in a job definition field, the server submits the job with the %VAR statement unresolved.

This command has the following format:

```
deletevar name("name") [context("context")]
```

name("*name*")

Specifies the name of the global variable. This name is not case sensitive.

Limits: 1-128 alphanumeric or underscore characters. The first character cannot be a number.

context("*context*")

(Optional) Specifies the name of the context that the global variable belongs to.

Limits: 1-128 alphanumeric or underscore characters

Default: DEFAULT context

Examples: "payroll", “01_group”

Example: Delete a Global Variable

This example deletes a global variable named statusmsg from the DEFAULT context:

```
deletevar name("statusmsg")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

deletevarctx Command—Delete a Global Variable Context

You can delete a global variable context if the context is no longer in use. Deleting a context also deletes all of its global variables. To delete a context, issue the DELETEVARCTX command.

Note: If you delete a global variable that is specified in a %VAR statement in a job definition field, the server submits the job with the %VAR statement unresolved.

This command has the following format:

```
deletevarctx context("context")
```

context("*context*")

Specifies the name of the context that the global variable belongs to.

Limits: 1-128 alphanumeric or underscore characters

Examples: "payroll", “01_group”

Note: To delete the context, you must have Alter permission for all of the global variables in the specified context or for the context itself.

Example: Delete a Global Variable Context

Suppose that you have Alter permission for the global variable context named 01_group. This example deletes the 01_group context:

```
deletevarctx context("01_group")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

dropvardep Command—Drop Variable Dependencies

You can drop some or all of a job's variable dependencies before the job becomes eligible for submission if the job no longer needs to wait for variable dependencies, or you want the job to become eligible for submission sooner. To drop variable dependencies, issue the DROPVARDEP command.

Notes:

- You can drop a dependency only for a running Application.
- To verify that the variable dependencies are dropped, issue the EVALVARDEP command to list the job's remaining variable dependencies.

This command has the following format:

```
dropvardep appl("appl") gen(gen) jobname("jobname") [qualifier("qualifier")]
drop(all) | drop(name("name") context("context") operator("operator")
value("value"))
```

appl("*appl*")

Specifies the name of the Application that the job belongs to.

Limits: 1-128 characters

Examples: "calcpay", "_process", “01prod”

gen(*gen*)

Specifies the Application generation number.

jobname("*jobname*")

Specifies the name of the job.

Limits: 1-128 characters

Examples: "job01", "_job1", “01_job”

qualifier("*qualifier*")

(Optional) Specifies the job qualifier.

Limits: 1-128 characters

Examples: "unix01", "_unix1", “01_unix”

drop(all) | drop(name("*name*") context("*context*") operator("*operator*") value("*value*"))

Specifies whether to drop all of the job's variable dependencies or to drop a specific variable dependency. Options are the following:

drop(all)

Indicates that all of the job's variable dependencies will be dropped.

drop(name("*name*") context("*context*") operator("*operator*") value("*value*"))

Indicates that a specific variable dependency will be dropped.

name("*name*")

Specifies the name of the global variable. This name must be unique in its variable context and is case-sensitive.

Limits: 1-128 alphanumeric or underscore characters. The first character cannot be a number.

context("*context*")

Specifies the name of the context that the global variable belongs to.

Limits: 1-128 alphanumeric or underscore characters.

operator("*operator*")

Specifies the expression operator. Options are =, !=, >, <, >=, <=, exist, !exist, contains, !contains.

value("*value*")

Specifies the value of the global variable.

Limits: Up to 1024 characters

Example: Drop All of a Job's Variable Dependencies

Suppose that generation 5 of the _calctotal Application is running. The Application contains a job named job01 that is in a VARWAIT state. The following example drops all of job01's variable dependencies.

```
dropvardep appl("_calctotal") gen(5) jobname("job01") drop(all)
```

The following command verifies that all of the job's variable dependencies were dropped:

```
evalvardep appl("_calctotal") gen(5) jobname("job01")
Could not find variable dependencies for job <_CALCTOTAL.5.JOB01>.
Either the job does not exist, or the job is not waiting for variable dependencies
to be satisfied.
```

Examples: Drop Individual Variable Dependencies

Suppose that generation 6 of the _calctotal Application is running. The Application contains a job named job02.initial that is in a VARWAIT state. The following example evaluates job02.initial's variable dependencies:

```
evalvardep appl("_calctotal") gen(6) jobname("job02”) qualifier(“initial")
Variable Dependencies:

([PAYROLL<paymonth> = "May"] &
 [PAYROLL<review> exists] &
 [PAYROLL<total> >= "1000"])

Evaluation:

[PAYROLL<paymonth> = "May"] = false
 [PAYROLL<review> exists] = false
 [PAYROLL<total> >= "1000"] = false
```

The following commands drop two of job02.initial's variable dependencies:

```
dropvardep appl("_calctotal") gen(6) jobname("job02”) qualifier(“initial")
drop(name("REVIEW") context("payroll") operator("exists") “”)
Command submitted.
dropvardep appl("_calctotal") gen(6) jobname("job02”) qualifier(“initial")
drop(name("TOTAL") context("payroll") operator(">=") value(“1000”))
```

The following command verifies that the two variable dependencies are dropped:

```
evalvardep appl("_calctotal") gen(6) jobname("job02”) qualifier(“initial")
Variable Dependencies:

([PAYROLL<paymonth> = "May"])

Evaluation:

[PAYROLL<paymonth> = "May"] = false
```

More information:

Issue a Command in the CLI Perspective (see page 10)

evalvardep Command—Evaluate Variable Dependencies

When a job is in a VARWAIT state, you can evaluate the job's variable dependencies. Evaluating the dependencies is helpful when you want to check the status of the job and the expressions that are waiting to be met before the job can be submitted. To evaluate a job's variable dependencies, issue the EVALVARDEP command.

This command has the following format:

```
evalvardep appl("appl") gen(gen) jobname("jobname") [qualifier("qualifier")]
```

appl("*appl*")

Specifies the name of the Application that the job belongs to.

Limits: 1-128 characters

Examples: "calcpay", "_process", “01prod”

gen(*gen*)

Specifies the Application generation number.

jobname("*jobname*")

Specifies the name of the job.

Limits: 1-128 characters

Examples: "job01", "_job1", “01_job”

qualifier("*qualifier*")

(Optional) Specifies the job qualifier.

Limits: 1-128 characters

Examples: "unix01", "_unix1", “01_unix”

Example: Evaluate a Job's Variable Dependencies

Suppose that generation 3 of the processapp Application is running. The Application contains a job named 01_job that is in a VARWAIT state. The following example evaluates the job's variable dependencies. The response shows the variable dependency expression statusmsg=”Calculation complete” is waiting to be met.

```
evalvardep appl("processapp") gen(3) jobname("01_job")
Variable Dependencies:

([PAYROLL<statusmsg> = "Calculation complete"] &
 [PAYROLL<paymonth> = "may"])

Evaluation:

[PAYROLL<statusmsg> = "Calculation complete"] = false
[PAYROLL<paymonth> = "may"] = true
current variable value = xxx
```

More information:

Issue a Command in the CLI Perspective (see page 10)

incrementvar Command—Increment an Integer-valued Global Variable

You can increment a global variable value by 1 if the value is an integer. To increment a global variable value, issue the INCREMENTVAR command.

Note: If the specified global variable does not exist, the global variable is created and its value is set to 1. If the specified global variable is already assigned the maximum integer value (2^{63}-1), its value is set to 1. If the value of the specified global variable is not an integer, you get an error.

The command has the following format:

```
incrementvar name("name") [context("context")]
```

name("*name*")

Defines the name of the global variable. This name must be unique in its variable context. This name is not case sensitive.

Limits: 1-128 alphanumeric or underscore characters. The first character cannot be a number.

context("*context*")

(Optional) Defines the name of the context that the global variable belongs to.

Limits: 1-128 alphanumeric or underscore characters

Default: DEFAULT context

Examples: "payroll", “01_group”

Example: Increment an Integer-valued Global Variable

Suppose that a global variable named quota in a payroll context contains a value of 10. The following example increments the value of the quota variable by 1:

```
incrementvar name("quota") context("payroll")
Command successful.
Value=11
```

More information:

Issue a Command in the CLI Perspective (see page 10)

listvar Command—List the Global Variables

You can view the available global variables by issuing the LISTVAR command.

This command has the following format:

```
listvar [name("name")] [context("context")] [verbose(true|false)]
```

name("*name*")

(Optional) Defines a filter to limit the variables displayed. You can use the asterisk (*) as a wildcard for zero or more characters and the question mark (?) as a wildcard for a single character. This name is not case sensitive.

Default: All variables

Example: b* searches all variable names that begin with b

context("*context*")

(Optional) Defines a filter to limit the contexts searched. You can use the asterisk (*) as a wildcard for zero or more characters and the question mark (?) as a wildcard for a single character. This filter is not case sensitive.

Default: All contexts

Example: s* searches all contexts that begin with s

verbose(true|false)

(Optional) Specifies whether to display details. Options are the following:

true

Displays details. Details include when the variable was created or last modified and who created the variables.

false

Does not display details.

Default: verbose(false)

Example: List the Global Variables

This example lists all of the global variables in all contexts. Details are displayed.

```
listvar name("*") context("*") verbose(true)
Context: DEFAULT

    statusmsg = final amount
    Created: Wed Aug 08 16:14:56 EDT 2007
    Created By: SCHEDMASTER
    Last Modified: Wed Aug 08 16:14:56 EDT 2007
    Last Modified By: SCHEDMASTER

Context: PAYROLL

    dept01 = hr
    Created: Wed Aug 08 15:35:46 EDT 2007
    Created By: USER1
    Last Modified: Wed Aug 15 12:49:43 EDT 2007
    Last Modified By: USER1

    dept02 = dev
    Created: Wed Aug 08 15:50:20 EDT 2007
    Created By: USER2
    Last Modified: Wed Aug 15 12:50:20 EDT 2007
    Last Modified By: USER2
```

More information:

Issue a Command in the CLI Perspective (see page 10)

listvarctx Command—List the Global Variable Contexts

You can view the available global variable contexts by issuing the LISTVARCTX command.

This command has the following format:

```
listvarctx
```

Example: List the Global Variable Contexts

This example lists all global variable contexts:

```
listvarctx
DEFAULT
PAYROLL
```

More information:

Issue a Command in the CLI Perspective (see page 10)

setvar Command—Create a Global Variable

You can create a global variable using the CLI. Global variables store information that you can reuse across Applications. To create a global variable, issue the SETVAR command.

This command has the following format:

```
setvar name("name") value("value") [context("context")]
```

name("*name*")

Defines the name of the global variable. This name must be unique in its variable context. This name is not case sensitive.

Limits: 1-128 alphanumeric or underscore characters. The first character cannot be a number.

value("*value*")

Defines the value of the global variable.

Limits: 1-1024 characters

Examples: "dept01", “final cost”, “20”, “01num”, "name@company.com"

context("*context*")

(Optional) Defines the name of the context that the global variable belongs to.

Limits: 1-128 alphanumeric or underscore characters

Default: DEFAULT context

Note: If the context does not exist, it will be created.

Example: Create a Global Variable

This example creates a global variable named dept in the payroll context. The variable is assigned the value of dept0102.

```
setvar name("dept") value("dept0102") context("payroll")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

setvar Command—Modify a Global Variable

You can modify an existing global variable if you need to change the variable's details. To modify a global variable, issue the SETVAR command.

Note: If the specified global variable name or context does not exist, a new variable is created with the specified information.

This command has the following format:

```
setvar name("name") value("value") [context("context")]
```

name("*name*")

Specifies the name of the global variable. This name is not case sensitive.

Limits: 1-128 alphanumeric or underscore characters. The first character cannot be a number.

value("*value*")

Defines the value of the global variable.

Limits: 1-1024 characters

Examples: "dept01", “final cost”, “20”, “01num”, "name@company.com"

context("*context*")

(Optional) Specifies the name of the context that the global variable belongs to.

Limits: 1-128 alphanumeric or underscore characters

Default: DEFAULT context

Example: Modify a Global Variable

This example modifies the value of a global variable named STATUS in the payroll context. The variable is assigned the value of “Not started”.

```
setvar name(“STATUS”) value(“Not started”) context("payroll")
```

More information:

Issue a Command in the CLI Perspective (see page 10)

Chapter 8: Work with Resources

This section contains the following topics:

adjustresourceproperty Command—Change the Value of a Resource Property by a Specified Amount

You can change the value of a resource property by a specified amount by issuing the ADJUSTRESOURCEPROPERTY command. The command can increase or decrease the number of available units of the resource. For a renewable resource, the command can also increase or decrease the maximum number of available units.

This command has the following format:

`adjustresourceproperty name("name") property("property") value(value)`

name("*name*")

Specifies the name of the resource.

property(*"property"*)

Specifies the resource property to be changed. Options are the following:

availability

Indicates the number of available units of the resource.

maximumavailability

Indicates the maximum number of available units of the resource. Applies to renewable resources only. Changing the maximum availability count also changes the availability count by the same amount. For example, if you increase the maximum availability count from three to five, the availability count increases by two.

value(*value*)

Changes the availability count of the resource by the specified amount. To increase the count, specify a positive integer. To decrease the count, specify a negative integer.

Limits: Any integer

Note: For renewable resources, if the availability count exceeds the maximum availability count, the availability count is set to the maximum availability count.

Example: Increase the Availability Count of a Resource

Suppose that the RES1 resource has an availability count of 6. The following example increases the number of available units of the resource to 9:

```
adjustresourceproperty name("RES1") property("availability") value(3)
```

Example: Decrease the Maximum Availability Count of a Resource

Suppose that the RES1 resource has an availability count of 7 and a maximum availability count of 10. The following example decreases the maximum number of available units of the resource to 6. Since the maximum number of units decreased by 4, the availability count is also decreased by 4 and changed to 3.

```
adjustresourceproperty name("RES1") property("maximumavailability") value(-4)
```

More Information:

Issue a Command in the CLI Perspective (see page 10)

createresource Commnad—Create a Resource

You must create a resource before you can set up a resource dependency in a job. You can create a resource by issuing the CREATERESOURCE command.

This command has the following format:

```
createresource name("name") type("type") [avail(avail)] [maxavail(maxavail)]
[description("description")]
```

name("*name*")

Specifies the name of the resource to be created.

type("*type*")

Identifies the type of resource to be created. Options are the following:

- depletable—Creates a depletable resource
- renewable—Creates a renewable resource
- threshold—Creates a threshold resource

avail(*avail*)

(Optional) Identifies the number of available units of the resource.

Default: 1

maxavail(*maxavail*)

(Optional) Identifies the maximum number of available units of the resource.

Note: You can apply this parameter only to renewable resources.

Default: 1

description("*description*")

(Optional) Identifies the description of the resource.

Example: Create a Renewable Resource

The following example creates a renewable resource named TEST_RES with availability of 5, maximum availability of 15, and description of “for test”:

```
createresource name ("TEST_RES") type("renewable") avail(5) maxavail(15)
description("for test")
```

Note: For more information about creating resources, see the *Services Perspective Help*.

More information:

Issue a Command in the CLI Perspective (see page 10)

deleteresource Command—Delete a Resource

You can delete a resource if it is no longer in use by issuing the DELETERESOURCE command.

Note: If a job requires the resource, the job goes into a RESWAIT state. The job remains in the RESWAIT state until the resource dependency is dropped or a new resource is created with the same name as the deleted resource.

This command has the following format:

```
deleteresource name("name")
```

name("*name*")

Specifies the name of the resource to be deleted.

Example: Delete a Resource

The resource is deleted immediately even if there are jobs using the resource. The following example deletes a resource named TEST_RES:

```
deleteresource name("TEST_RES")
```

Note: For more information about deleting resources, see the *Services Perspective Help*.

More information:

Issue a Command in the CLI Perspective (see page 10)

listresources Command—List Resources

You can view a list of existing resources to view their details. You can view the list of resources by issuing the LISTRESOURCES command.

This command has the following format:

```
listresources [name("name")]
```

name("*name*")

(Optional) Specifies the name of the resources to be listed. Enter the complete or partial resource name in the Command field to limit the resources displayed.

Default: all resources

Note: You must include a wildcard for a partial name. For example, RES* displays all the resources whose names start with RES.

Example: List Resource

The following example lists all the available resources defined in the server:

```
listresources
RES1
Renewable
Last Modified Time: Thu Sep 27 18:28:25 IST 2007
Last Modified By: SCHEDMASTER

RES2
Renewable
Last Modified Time: Thu Sep 27 18:28:36 IST 2007
Last Modified By: SCHEDMASTER
```

Note: For more information about listing resources, see the *Services Perspective Help*.

More information:

Issue a Command in the CLI Perspective (see page 10)

listresourcestatus Command—Display Resource Status

You can view the status of a resource and the number of times it has been used, reserved, or needed. You can view a resource's status by issuing the LISTRESOURCESTATUS command.

This command has the following format:

```
listresourcestatus name("name")
```

name("*name*")

Specifies the name of the resource to be listed.

Example: Display Resource Status

The following example lists the resource status for a specified resource:

```
listresourcestatus name("res2")
Resource RES2 Renewable
The resource is active.
Availability: 2
MaximumAvailability: 10
Used: 3
    3 used by UNIX0, application VER.3
Reserved: 0
Needed: 0
```

Note: For more information about displaying resource status, see the *Services Perspective Help*.

More information:

Issue a Command in the CLI Perspective (see page 10)

resetresourceproperty Command—Reset the Value of a Resource Property

You can reset the value of a resource property by issuing the RESETRESOURCEPROPERTY command. The command can increase or decrease the number of available units of the resource. For a renewable resource, the command can also increase or decrease the maximum number of available units.

This command has the following format:

```
resetresourceproperty name("name") property("property") [value(value)]
```

name("*name*")

Specifies the name of the resource.

property(*"property"*)

Specifies the resource property to be reset. Options are the following:

availability

Indicates the number of available units of the resource.

maximumavailability

Indicates the maximum number of available units of the resource. Applies to renewable resources only. Changing the maximum availability count also changes the availability count by the same amount. For example, if you increase the maximum availability count from three to five, the availability count increases by two.

value(*value*)

(Optional) Sets the availability count of the resource to the specified value.

Default: 0

Limits: Any integer 0 or greater

Note: For renewable resources, if the availability count exceeds the maximum availability count, the availability count is set to the maximum availability count.

Example: Reset the Availability Count of a Resource

The following example resets the number of available units of the RES1 resource to 79:

```
resetresourceproperty name("RES1") property("availability") value(79)
```

Example: Reset the Maximum Availability Count of a Resource

Suppose that the RES1 resource has an availability count of 79 and a maximum availability count of 80. The following example resets the maximum number of available units of the resource to 100. Since the maximum number of units increased by 20, the availability count is also increased by 20 and reset to 99.

```
resetresourceproperty name("RES1") property("maximumavailability") value(100)
```

More Information:

Issue a Command in the CLI Perspective (see page 10)

resourcemanagerdump Command—Display Resource Details

For debugging purposes, you can display details about the resources defined on the server and the processed resource requests (orders). You can view the resource and order details by issuing the RESOURCEMANAGERDUMP command.

This command has the following format:

```
resourcemanagerdump
```

Example: Display Resource Details

This example shows part of a response to the RESOURCEMANAGERDUMP command:

```
resourcemanagerdump
Resource manager information:
Resource:
    CybResource: [name:RENEWABLE][maxAvailability:8][availability:5]
[reservedCount:0][useCount:0]reservedDetailMap[]
    CybResource: [name:DEPLETABLE][maxAvailability:-1][availability:1]
[reservedCount:0][useCount:0]reservedDetailMap[]
    CybResource: [name:THRESHOLD][maxAvailability:-1][availability:77]
[reservedCount:0][useCount:0]reservedDetailMap[]
Resource order:
    CybResourceOrder[[requesterId=THRESHOLD.226.RWNT.3][orderId=10580]
[absorb=true][priority=3][requestTime=Wed Sep 17 14:32:01 EDT 2009]
[satisfiedTime=null]resourceOrderItemMap[CybResourceOrderItem:
[resourceName:UNDEFINED][quantity:30]]]
...
```

More information:

Issue a Command in the CLI Perspective (see page 10)

updateresourcedefinition Command—Update a Resource Definition

You can update an existing resource's definition by issuing the UPDATERESOURCEDEFINITION command.

Note: You must cold start the server for the new value to take effect.

This command has the following format:

```
updateresourcedefinition name("name") [avail(avail)] [maxavail(maxavail)]
[description("description")]
```

name("*name*")

Specifies the name of the resource to be updated.

avail(*avail*)

(Optional) Defines the number of available units of the resource.

maxavail(*maxavail*)

(Optional) Defines the maximum number of available units of the resource. Applies to renewable resources only.

description("*description*")

(Optional) Defines a description of the resource.

Example: Update a Resource Definition

The following example updates the resource definition of the res1 resource. The command resets the maximum number of available units to 9, the number of available units to 6, and the description to “resource description”. The changes take effect when the server is cold-started:

```
updateresourcedefinition name("resource1") avail(6) maxavail(9)
description("resource description")
```

Note: For more information about updating resources, see the *Services Perspective Help*.

More information:

Issue a Command in the CLI Perspective (see page 10)

Index

A

B

C

D

E

F

G

H

I

J

L

M

P

Q

R

S

T

U

V