CA Workload Automation AE

User Guide

r11.3

technologies

CA Technologies Product References

This document references the following CA Technologies products:

- CA Access Control
- CA AutoSys Workload Automation Connect Option (CA AutoSys WA Connect Option)
- CA Embedded Entitlements Manager (CA EEM)
- CA Job Management Option
- CA Jobtrac™ Job Management (CA Jobtrac JM)
- CA Network and Systems Management (CA NSM)
- CA NSM Event Management
- CA NSM Management Command Center (CA NSM MCC)
- CA Scheduler® Job Management (CA Scheduler JM)
- CA Service Desk
- CA Spectrum Automation Manager (formerly named CA DCA Manager)
- CA Universal Job Management Agent (CA UJMA)
- CA Workload Automation AE (formerly named CA AutoSys Workload Automation)
- CA Workload Automation Agent for UNIX (CA WA Agent for UNIX)
- CA Workload Automation Agent for Linux (CA WA Agent for Linux)
- CA Workload Automation Agent for Windows (CA WA Agent for Windows)
- CA Workload Automation Agent for i5/OS (CA WA Agent for i5/OS)
- CA Workload Automation Agent for Application Services (CA WA Agent for Application Services)
- CA Workload Automation Agent for Web Services (CA WA Agent for Web Services)
- CA Workload Automation Agent for Databases (CA WA Agent for Databases)
- CA Workload Automation Agent for SAP (CA WA Agent for SAP)
- CA Workload Automation Agent for PeopleSoft (CA WA Agent for PeopleSoft)
- CA Workload Automation Agent for Oracle E-Business Suite (CA WA Agent for Oracle E-Business Suite)
- CA Workload Automation Agent for z/OS (CA WA Agent for z/OS)
- CA Workload Automation EE (formerly named CA ESP Workload Automation)
- CA Workload Automation SE (formerly named CA 7 Workload Automation)

- CA Workload Control Center (CA WCC)
- CA Desktop and Server Management (CA DSM)

Contact CA Technologies

Contact CA Support

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

- 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

Provide Feedback

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

If you would like 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: Introduction

This section contains the following topics:

Intended Audience

This document is for schedulers and operators who define, schedule, monitor, and control workload using CA Workload Automation AE. The concepts in this document can also help application programmers or developers who create custom applications to work with CA Workload Automation AE.

This *User Guide* is a supplement to the *Reference Guide* and provides overview topics and concepts. It also contains step-by-step procedures to help you define basic jobs.

The *Reference Guide* provides detailed information about commands, attributes, and JIL (Job Information Language) syntax.

To use this document, you must be familiar with the operating system(s) where the jobs run and any third-party products or software technology that the jobs use.

Notes:

- The UNIX instructions in this document also apply to Linux systems unless otherwise noted.
- The term *Windows* refers to any Microsoft Windows operating system supported by CA Workload Automation AE unless otherwise noted.
- For information about the CA Workload Automation AE components and how they interact, see the *UNIX Implementation Guide* or *Windows Implementation Guide*.

Automated Job Control

CA Workload Automation AE is an automated job control system for scheduling, monitoring, and reporting.

A *job* is any single command, executable, script, or batch file. These jobs can reside on any configured machine that is attached to a network. Corresponding *job definitions* contain attributes that define the job properties, including the conditions specifying when and where a job should run.

As with most control systems, there are many ways to correctly define and implement jobs. It is likely that the way you use CA Workload Automation AE to address your distributed computing needs will evolve over time. As you become more familiar with the product features and the characteristics of your jobs, you can refine your use of CA Workload Automation AE.

Agents and Agent Plug-ins

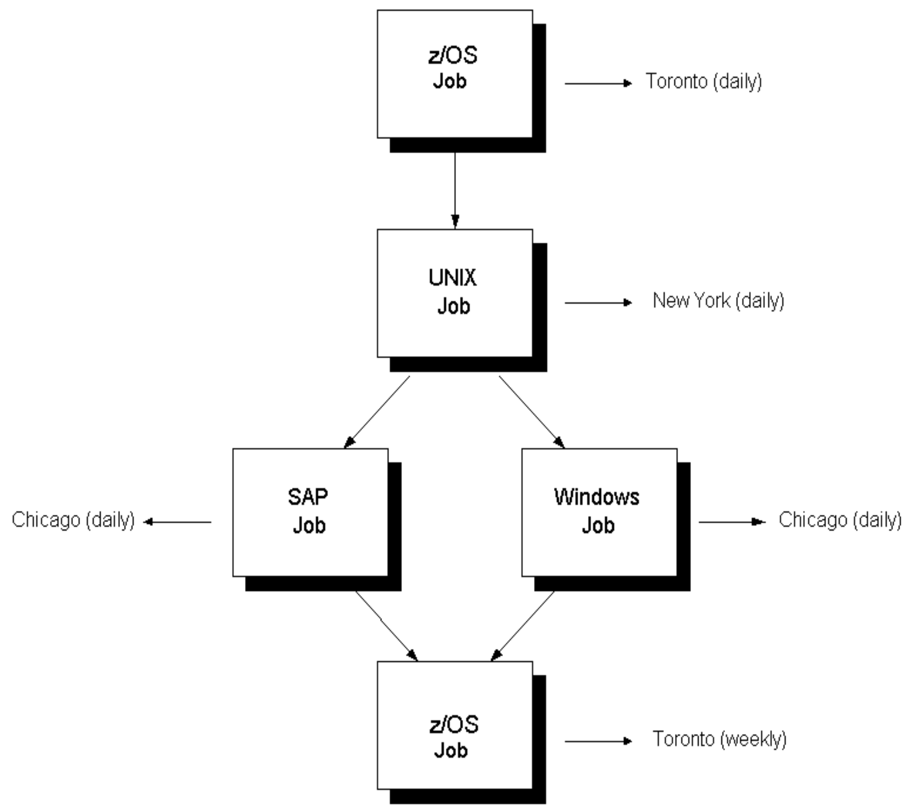
Agents are the key integration components of CA workload automation products. Agents let you automate, monitor, and manage workload on all major platforms, applications, and databases. To run workload on a particular system, you install an agent on that system. If your workload must run on a UNIX computer, for example, you can install and configure the CA WA Agent for UNIX. The agent lets you run UNIX scripts, execute UNIX commands, transfer files using FTP, monitor file activity on the agent computer, and perform many other tasks.

You can extend the functionality of the agent by installing one or more agent plug-ins in the agent installation directory. If you have a relational database such as Oracle, for example, you can install a database agent plug-in to query and monitor the database. Other agent plug-ins are also available. For more information, see the *Implementation Guide* for the appropriate agent plug-in.

Note: The agent plug-ins are only available for UNIX, Linux, and Windows operating environments.

Example: Workload with Different Types of Jobs

The following workload contains z/OS jobs, a UNIX job, an SAP job, and a Windows job, running on different computers, in different locations, and at different times:



agentparm.txt File

The agent is installed with a configuration file named agentparm.txt. Some of the parameters in this file control the properties and behavior of job types. Your agent administrator can add and modify the parameters as required.

Note: For more information about the agentparm.txt file, see the *UNIX Implementation Guide, Windows Implementation Guide*, and *CA Workload Automation Agent for UNIX, Linux, or Windows Implementation Guide.*

Legacy Agent Replaced by CA Workload Automation Agent

The new CA Workload Automation Agent for UNIX, Linux, or Windows replaces the Remote Agent (auto_remote) that was provided with Unicenter AutoSys JM r4.5 and r11. The r11.3 documentation refers to auto_remote as the *legacy agent*.

The new agent provides additional job types, including monitoring and FTP jobs. The agent is automatically installed on the computer where CA Workload Automation AE is installed. You can also install the agent on remote computers to run jobs on those computers.

Jobs

In the CA Workload Automation AE environment, a job is a single action that can be performed on a valid agent computer. For example, on UNIX, you can run a script, issue a command, transfer files using FTP, and monitor file activity and processes on the agent computer. Similarly, on Windows, you can run an executable or batch file, issue a command, transfer files using FTP, and monitor files or processes. To run a job, you must create a job definition. A job definition includes attributes that define what the job does, when it runs, and where it runs.

Events

CA Workload Automation AE is completely event-driven; for the scheduler to activate a job, an event must occur on which the job depends. For example, a job can be activated when a prerequisite job has completed running successfully or a required file has been received.

Events can come from a number of sources, including:

- Jobs changing states, such as starting, finishing successfully, and so on.
- Internal verification agents, such as detected errors.
- Events sent with the sendevent command, from the CA WCC GUI, or from user applications.

As each event is processed, the scheduler scans the database for jobs that are dependent on that event. If the event satisfies another job's starting conditions, that job is either started immediately or queued for the next qualified and available computer. The completion of one job can cause another job to start so that jobs progress in a controlled sequence.

Alarms

Alarms are special events that notify operators of situations requiring their attention. Alarms are integral to the automated use of CA Workload Automation AE. That is, you can schedule jobs to run based on a number of conditions, but some facility is necessary for addressing incidents that require manual intervention. For example, a set of jobs could depend on the arrival of a file, and the file may be long overdue. It is important that someone investigate the situation, make a decision, and resolve the problem.

The following are some important aspects of alarms:

- Alarms are informational only. Any action taken must be initiated by a separate action event.
- Alarms are system messages about a detected incident.
- Alarms are sent through the system as events.
- Alarms have special monitoring features to make sure they are noticed.

Utilities

CA Workload Automation AE uses its own specification language (JIL) and client utilities to help you define, control, and report on jobs. The JIL language is processed by the JIL client executable, which reads and interprets the JIL statements that you enter and then performs the appropriate actions.

CA Workload Automation AE also provides client programs for controlling and reporting on jobs. For example, the autorep command lets you generate a variety of reports about job execution, and the sendevent command lets you manually control job processing.

Additional utilities are provided to help you troubleshoot, run monitors and reports, and start and stop CA Workload Automation AE and its components. CA Workload Automation AE also provides a database maintenance utility that runs daily by default.

Commands

CA Workload Automation AE commands let you define, monitor, and manage workload, and configure and control the system.

The following commands start CA Workload Automation AE:

- as_server—Runs the application server.
- event_demon—Represents the binary (the scheduler) that runs CA Workload Automation AE.
- eventor—Starts the scheduler (the event demon). This command applies to UNIX only.

The following commands maintain databases:

- archive_events—Archives data from the database to a flat file.
- archive_jobs—Deletes obsolete job versions from the database.
- autotimezone—Queries, adds, and deletes entries in the ujo_timezones table.
- autotrack—Tracks changes, such as job definition changes, sendevent calls, and job overrides, to the database.
- clean_files—Deletes old agent log files. This command applies to the legacy agent only.
- DBMaint—Runs the dbstatistics and archive_events commands to perform maintenance on the CA Workload Automation AE database.
- dbspace—Calculates and reports the disk space used by the CA Workload Automation AE database tables.
- dbstatistics—Generates statistics in the data servers to maintain an optimal performance environment.

The following commands manage security:

- as_config—Manages the encryption keys for the application server and for CA Workload Automation Agent for UNIX, Linux, or Windows.
- as_safetool—Maintains authentication certificates and installs or removes CA EEM security policies.
- autosec_test—Simulates a call to the security subsystem.
- autosys_secure—Defines security settings.

The following commands check system status:

- as_info—Returns installation information to standard output.
- autoaggr—Aggregates data into hourly, daily, weekly, and monthly tables that other programs use to generate reports.
- autoflags—Returns system information.
- autoping—Verifies server and agent communication.
- chk_auto_up—Determines whether the application server, scheduler, and event server are running.
- time0—Calculates internal CA Workload Automation AE time.

The following commands manage assets, such as jobs, machines, and calendars:

- astail—Displays the last 10 lines of a file. This command applies to UNIX only.
- autocal_asc—Adds, deletes, prints, exports, and imports calendars.
- autoprofm—(Windows only) Converts your profiles from a previous release of CA Workload Automation AE to a file format that works with the CA Workload Automation Agent.
- cron2jil—Converts UNIX crontab data to a JIL script or calendar file. This command applies to UNIX only.
- jil—Runs the Job Information Language processor that interprets the subcommands that add, update, and delete jobs, machines, monitors, and browsers (reports).

The following commands monitor jobs and report job status:

- autorep—Reports information about jobs, user-defined job types, external instances, machines, virtual resources, and global variables defined in the database.
- autostatad—Reports the status of a CA AutoSys Workload Automation Adapter job to standard output.
- autostatus—Reports the status of a job or the value of a global variable to standard output.
- chase—Determines which jobs are in a STARTING or RUNNING state. The chase command also verifies the agent is running.
- forecast—Reports future job flow.
- job_depends—Generates detailed reports of job dependencies and conditions.
- monbro—Runs a monitor or report and returns the results to standard output.
- autosyslog—Displays the scheduler, application server, or agent log file for a specified job.

The following command sends event commands:

- sendevent—Sends events to the scheduler.

The following command provides integration with CA Service Desk:

- auto_svcdesk—Opens CA Service Desk tickets on behalf of a job or an action that may occur on CA Workload Automation AE.

Note: For more information about the syntax of a command, see the *Reference Guide*.

Issue a Command on UNIX

You issue commands to define, monitor, and manage workload. You can also issue commands to configure and control CA Workload Automation AE. For example, you can issue the jil command to define jobs and the autosys_secure command to define security settings.

Note: The CA Workload Automation AE client must be installed on the computer where you want to issue commands. Before you can issue a command, ensure that your administrator has completed the following:

- Run the autosys.*shellname*.*mymachine* script for your shell program to source the environment.
- Defined your UNIX user ID and password on CA Workload Automation AE.

To issue a command on UNIX

1. Run the shell that is sourced to use CA Workload Automation AE.
2. Enter the command at the UNIX operating system prompt.

   The command is issued.

Note: For more information about CA Workload Automation AE commands and their syntax, see the *Reference Guide*. You can also view the help for the commands by using the UNIX man command. For example, to view the reference page for the sendevent command, enter **man sendevent** at the command prompt. If you do not see the sendevent man page, check that the MANPATH environment variable is set correctly. This variable is usually set in the $AUTOUSER/autosys.*shell.hostname* files (for example, $AUTOUSER/autosys.ksh.myhostname). These files set up the CA Workload Automation AE environment.

Issue a Command on Windows

You issue commands to define, monitor, and manage workload. You can also issue commands to configure and control CA Workload Automation AE. For example, you can issue the jil command to define jobs and the autosys_secure command to define security settings.

Note: The CA Workload Automation AE client must be installed on the computer where you want to issue commands. The client installs the CA Workload Automation AE Command Prompt, which is required to run commands. You cannot use the MS-DOS command prompt.

To issue a command on Windows

1. Click Start, All Programs, CA, Enterprise Workload Automation, Command Prompt(*instance_name*).

   The CA Workload Automation AE command prompt opens. The command prompt presets all the environment variables for the instance.

2. Enter the command.

   The command is issued.

Security

CA Workload Automation AE includes features that let you secure objects such as jobs, calendars, cycles, global variables, machines, and resources. You can delegate administrative privileges to these objects to specific users or user groups.

For more information about data encryption and system-level, native, or external security, see the *CA Workload Automation Security Guide*.

For additional information about configuring encryption between CA Workload Automation AE and agents, see the *UNIX Implementation Guide* and *Windows Implementation Guide*.

Chapter 2: Working with JIL

This section contains the following topics:

The jil Command and JIL (Job Information Language)

Job Information Language (JIL) is a scripting language that lets you define and modify assets such as jobs, global variables, machines, job types, external instances, and blobs.

The jil command runs the language processor that intepretes JIL. You use JIL subcommands and attributes to specify the asset definitions that you want to create or modify. The jil command loads the data that you specify into the database.

You can define and modify assets using the following methods:

- Issue the jil command, which displays the JIL command prompt. You enter subcommands interactively one line at a time. When you exit the command prompt, the language processor interprets the asset definition and loads it into the database.
- Create a JIL script by entering subcommands and attribute statements in a text file. You redirect the JIL script to the jil command. The jil command activates the language processor, interprets the information in the script, and loads this information in the database.

Note: You can also submit job definitions using CA WCC. For more information about using CA WCC to define jobs, see the *CA Workload Control Center Online Help*.

JIL Subcommands

JIL subcommands lets you define and modify asset definitions. You specify JIL subcommands using the jil command.

The following JIL subcommands define and modify jobs and boxes:

delete_box

Deletes an existing box job and all the jobs in that box from the database.

delete_job

Deletes a job from the database. If the specified job is a box job, the box job is deleted and the jobs in the box become stand-alone jobs.

insert_job

Adds a new job definition to the database.

override_job

Defines a one-time override for an existing job definition. This override affects the job for the next run only.

update_job

Updates an existing job definition in the database.

The following JIL subcommands define and modify machines:

delete_machine

Deletes an existing real or virtual machine definition from the database.

insert_machine

Inserts a new real or virtual machine definition in the database. A machine must be defined before it can be used in a job definition.

update_machine

Updates an existing machine in the database.

The following JIL subcommands define and modify monitor or report definitions:

delete_monbro

Deletes the specified monitor or report definition from the database.

insert_monbro

Adds a new monitor or report definition to the database.

update_monbro

Updates an existing monitor or report definition in the database.

The following JIL subcommands define and modify job types:

delete_job_type

Verifies that no jobs are currently defined with the specified job type, then deletes the specified job type definition from the database.

insert_job_type

Adds a new user-defined job type definition to the database. This is the only way to create a user-defined job type.

update_job_type

Updates an existing user-defined job type definition in the database. You can use update_job_type to change the values of the command and description attributes.

The following JIL subcommands define and modify blobs and globs:

insert_blob

Adds a new blob definition associated with an existing job.

insert_glob

Adds a new glob definition referenced by a given name.

delete_blob

Decouples a blob definition from an existing job and deletes the blob from the database.

delete_glob

Deletes the specified glob definition from the database.

Note: Blobs and globs can only contain the following characters: A-Z, a-z, and 0-9.

The following JIL subcommands define and modify external instances:

delete_xinst

Deletes the specified external instance definition from the database.

insert_xinst

Adds a new external instance definition to the database.

update_xinst

Updates an existing external instance definition in the database.

The following JIL subcommands define and modify resources:

delete_resource

Deletes a virtual resource from the database.

insert_resource

Adds a new virtual resource definition to the database.

update_resource

Updates an existing virtual resource definition in the database.

JIL Syntax Rules

JIL scripts contain one or more JIL subcommands and one or more attribute statements; these elements constitute a job definition. When writing a JIL script, you must follow these syntax rules:

Rule 1

Each subcommand uses the following form:

```
sub_command:object_name
```

sub_command

Defines a JIL subcommand.

object_name

Defines the name of the object (for example, a job or a machine) to act on.

Rule 2

You can follow each subcommand with one or more attribute statements. These statements can occur in any order and are applied to the object specified in the preceding subcommand. A subsequent subcommand begins a new set of attributes for a different object. The attribute statements have the following form:

```
attribute_keyword:value
```

attribute_keyword

Defines a valid JIL attribute.

value

Defines the setting to apply to the attribute.

Notes:

- The entire *attribute_keyword*:*value* statement can be up to 4096 characters.
- The following subcommands do not accept attributes: delete_box, delete_job, delete_job_type, delete_xinst, delete_monbro, and delete_glob.

Rule 3

You can enter multiple attribute statements on the same line, but you must separate the attribute statements with at least one space.

Rule 4

A box job definition must exist before you can add jobs to it.

Rule 5

Valid *value* settings can include any of the following characters:

- Uppercase and lowercase letters (A-Z, a-z)
- Hyphens (-)
- Underscores (_)
- Pound signs (#)
- Numbers (0-9)
- Colons (:), if the colon is escaped with quotation marks (" ") or a preceding backslash (\)
- The at character (@)

Note: Object names can only contain the following characters: a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#). Do not include embedded spaces or tabs.

Rule 6

Because JIL parses on the combination of a keyword followed by a colon, you must use escape characters (a backslash) or enclose the value in quotation marks with any colons used in an attribute statement's value. For example, to define the start time for a job, specify 10\:00or “10:00”.

Note: When specifying drive letters in commands, you must use escape characters with the colon (:). For example, "C:\tmp" and C\:\tmp are valid; C:\tmp is not.

Rule 7

Use one of the following methods to indicate comments in a JIL script:

- To comment an entire line, put a pound sign (#) in the first column.
- To comment one or more lines, you can use the C programming syntax for beginning a comment with a forward slash and asterisk (/*) and ending it with an asterisk and a forward slash (*/). For example:

  ```
  /* this is a comment */
  ```

Rule 8

You can use the blob_input attribute to manually enter multiline text. This attribute is only valid for the insert_job, update_job, insert_blob, and insert_glob subcommands. The blob_input attribute has the following form:

```
blob_input:<auto_blobt> this is a multi-
line text
</auto_blobt>
```

Note: Use the auto_blobt meta-tags to indicate the beginning and end of multiline text. JIL interprets every character input between the auto_blobt meta-tags literally. This implies that JIL does not enforce any of the previously discussed rules for text entered in an open auto_blobt meta-tag.

Rule 9

To specify a comma in *keyword=value* combinations, you must do *one* of the following:

- Escape the comma with a backslash, as follows:

  ```
  j2ee_parameter: String=Hello1\, World
  ```

- Enclose the entire value in quotation marks, as follows:

  ```
  j2ee_parameter: String="Hello1, World"
  ```

If the *keyword=value* combination contains commas and quotation marks, you must escape the commas and quotation marks as follows:

```
j2ee_parameter: String=\"Hello1\, World\"
```

Issue JIL in Interactive Mode on UNIX

You can issue the jil command and subcommands on UNIX at the operating system prompt. The operating system prompt lets you enter data interactively. You can also create aliases for the commands that you use frequently. The interactive mode is helpful when you do not need to issue JIL commands in batch. For example, you can use the interactive mode when you want to test specific job definitions.

Note: Before you can issue commands at the operating system prompt, the CA Workload Automation AE environment must be sourced in the shell and your UNIX user ID and password must be defined on CA Workload Automation AE. For more information about sourcing the environment and defining user IDs, see the *UNIX Implementation Guide*.

To issue JIL in interactive mode on UNIX

1. Run the shell that is sourced to use CA Workload Automation AE.
2. Do *one* of the following:
   - Enter **jil** at the operating system prompt.

     The JIL command prompt is displayed for the local CA Workload Automation AE instance.
   - Enter **jil -S** *instance* at the operating system prompt.

     The JIL command prompt is displayed for the specified CA Workload Automation AE instance.

   instance

   Specifies the CA Workload Automation AE instance where you want to issue the JIL subcommand. For single-instance environments, the jil command defaults to the only available instance.
3. Enter a JIL subcommand and follow the prompts.
4. Enter **exit** when you are done entering the data.

   JIL interactive mode ends and the data is loaded into the database.

Issue JIL in Interactive Mode on Windows

You can issue the jil command and subcommands on Windows by using the CA Workload Automation AE instance command prompt. The instance command prompt lets you enter data interactively. The interactive mode is helpful when you do not need to issue JIL commands in batch. For example, you can use the interactive mode when you want to test specific job definitions.

To issue JIL in interactive mode on Windows

1. Click Start, Programs, CA, Workload Automation AE, Command Prompt (*instance_name*).

   The instance command prompt is displayed.
2. Do *one* of the following:
   - Enter **jil**.

     The JIL command prompt is displayed for the local CA Workload Automation AE instance.
   - Enter **jil -S** *instance*.

     The JIL command prompt is displayed for the specified CA Workload Automation AE instance.

instance

Specifies the CA Workload Automation AE instance where you want to issue the JIL subcommand. For single-instance environments, the jil command defaults to the only available instance.

3. Enter a JIL subcommand and follow the prompts.
4. Enter **exit** when you are done entering the data.

   JIL interactive mode ends and the data is loaded into the database.

Notes:

- You cannot use the MS-DOS command prompt to issue JIL subcommands.
- To issue commands that run on a UNIX computer from Windows, you must use the syntax appropriate to the operating system where the job runs.
- For the jil command to work properly, the correct environment variables must be assigned. For more information about environment variables, see the *Windows Implementation Guide*.
- For more information about the jil command, see the *Reference Guide*.

Issue JIL Using a Script on UNIX

You can embed JIL subcommands and attribute statements in a shell script and redirect the script to the jil command. The jil command activates the language processor, interprets the information in the script, and loads this information in the database. Using a script is helpful when you want to issue JIL subcommands in batch.

Note: Before you can run the JIL script, the CA Workload Automation AE environment must be sourced in the shell and your UNIX user ID and password must be defined on CA Workload Automation AE. For more information about sourcing the environment and defining user IDs, see the *UNIX Implementation Guide*.

To issue JIL using a script on UNIX

1. Run the shell that is sourced to use CA Workload Automation AE.
2. Do *one* of the following:
   - Enter **jil <** *script* at the operating system prompt.

     The subcommands in the JIL script are issued on the local CA Workload Automation AE instance and the data is loaded into the database.
   - Enter **jil -S** *instance* < *script* at the operating system prompt.

     The subcommands in the JIL script are issued on the specified CA Workload Automation AE instance and the data is loaded into the database.

script

Specifies the name of the script that contains the JIL subcommands and attributes.

instance

Specifies the CA Workload Automation AE instance where you want to run the JIL script. For single-instance environments, the jil command defaults to the only available instance.

Note: For more information about the jil command, see the *Reference Guide*.

Example: Submit a Job Definition Using a JIL Script

This example redirects a file called my_jil_script to the jil command on the PRD instance of CA Workload Automation AE.

```
jil –S PRD < my_jil_script
```

Issue JIL Using a Script on Windows

You can embed JIL subcommands and attribute statements in a script and redirect the script to the jil command. The jil command activates the language processor, interprets the information in the script, and loads this information in the database. Using a script is helpful when you want to issue JIL subcommands in batch.

To issue JIL using a script on Windows

1. Click Start, Programs, CA, Workload Automation AE, Command Prompt (*instance_name*).

   The instance command prompt is displayed.

2. Do *one* of the following:

   - Enter **jil** < *script*.

     The subcommands in the JIL script are issued on the local CA Workload Automation AE instance and the data is loaded into the database.

   - Enter **jil -S** *instance* < *script*.

     The subcommands in the JIL script are issued on the local CA Workload Automation AE instance and the data is loaded into the database.

script

Specifies the name of the script that contains the JIL subcommands and attributes.

instance

Specifies the CA Workload Automation AE instance where you want to run the JIL script. For single-instance environments, the jil command defaults to the only available instance.

Notes:

- You cannot use the MS-DOS command prompt to redirect a JIL script to the jil command.
- To issue commands that run on a UNIX computer from Windows, you must use the syntax appropriate to the operating system where the job runs.
- For the jil command to work properly, the correct environment variables must be assigned. For more information about environment variables, see the *Windows Implementation Guide*.
- For more information about the jil command, see the *Reference Guide*.

Example: Submit a Job Definition to the Local Instance Using a JIL Script

This example redirects a file called newjobdef to the jil command on the local instance of CA Workload Automation AE.

```
jil < newjobdef
```

Example JIL Script

The following JIL script creates a command job, a file watcher job, and a box job to do the following:

1. You expect a file named /DOWNLOAD/MAINFRAME/SALE.RAW to arrive from the mainframe some time after 2:00 a.m.
2. When the file arrives, the command file named filter_mainframe_info processes it, and the results are sent as an output to the file named /DOWNLOAD/MAINFRAME/SALES.SQL.
3. When the actions are complete, the SQL commands in the file named /DOWNLOAD/MAINFRAME/SALES.SQL runs.

```
# Example of a Machine
insert_machine: lowgate
type: a
# Example of Jobs
insert_job: Nightly_Download
job_type: box
date_conditions: yes
days_of_week: all
start_times: "02:00"
insert_job: Watch_4_file
job_type: ft
box_name: Nightly_Download
watch_file: /DOWNLOAD/MAINFRAME/SALES.RAW
watch_file_type: generate
machine: lowgate
insert_job: filter_data
job_type: cmd
box_name: Nightly_Download
condition: success(Watch_4_file)
command: filter_mainframe_info
machine: lowgate
std_in_file: /DOWNLOAD/MAINFRAME/SALES.RAW
insert_job: parse_data
job_type: cmd
box_name: Nightly_Download
condition: success(filter_data)
machine: lowgate
command: isql -U mutt -P jeff
std_in_file: /DOWNLOAD/MAINFRAME/SALES.SQL
std_out_file: /LOG/parse_data.out
std_err_file: /LOG/FilterMFLog.err/
```

Chapter 3: Working with Machines

This section contains the following topics:

Machines

Before you can schedule jobs to run on a machine, you must define the machine to CA Workload Automation AE. You can define real machines, virtual machines, or real machine pools.

Note: Do not define localhost (see page 41) using the insert_machine: localhost command. The localhost machine name is a reserved name.

Real Machines

A *real machine* is any single machine that meets the following criteria:

- It has been identified in the appropriate network so that CA Workload Automation AE can access it.
- An agent, CA AutoSys WA Connect Option, CA NSM, or CA UAJM is installed, which lets CA Workload Automation AE run jobs on it.
- It is defined to CA Workload Automation AE as a real machine using JIL.

A real machine must meet these conditions to run jobs. However, for CA Workload Automation AE to perform intelligent load balancing and queuing while running jobs, it must know the relative processing power of the real machines. CA Workload Automation AE uses the virtual machines to provide load balancing and queuing.

Virtual Machines

A v*irtual machine* is a machine definition that references one or more existing real machine definitions. By defining virtual machines to CA Workload Automation AE and submitting jobs to run on those machines, you can specify the following:

- Run-time resource policies (or constraints) at a high level.
- That CA Workload Automation AE automatically runs those policies in a multi-machine environment.

Note: Previous releases of CA Workload Automation AE required that all machines in a virtual machine be of the same type. In the current release, the component real machines in a virtual machine definition can be UNIX or Windows machines or a mix of both.

Real Machine Pools

A real machine pool is similar to a virtual machine in that it references one or more existing real machine definitions. Real machine pools are used specifically to integrate with CA Spectrum Automation Manager for load balancing, When a job is scheduled that references a real machine pool, the node names associated with the machines referenced are used by CA Spectrum Automation Manager to assign the real machine to run the job.

The localhost Definition

The localhost machine name is a reserved name. You cannot define a machine for localhost by creating an **insert_machine: localhost** definition.

By default, the localhost value is resolved to the name of the machine where the CA Workload Automation AE scheduler was installed. You can override the reserved localhost value to the name of another real machine using the local machine definition setting. On UNIX, you can configure this setting using the LocalMachineDefinition parameter in the configuration file. On Windows, you can configure this setting using the Local Machine Definition field in the Scheduler window of CA Workload Automation AE Administrator (autosysadmin).

You must create a machine definition in the database for the machine resolved from the localhost. To create a machine definition, use the insert_machine JIL command.

As part of the CA Workload Automation AE installation process, your administrator must have created a machine definition for the default localhost (the machine where the scheduler was installed) in the database. If you configure the local machine definition setting to another machine, you must create a definition for that machine in the database. For example, if you configure the local machine definition setting to a machine named prod, you must define machine prod using the **insert_machine: prod** command.

Note: For more information about the LocalMachineDefinition parameter in the configuration file (UNIX), see the *Administration Guide*. For more information about the Local Machine Definition (Windows), see the *Online Help* for CA Workload Automation AE Administrator.

How the localhost Value is Resolved

If the **machine: localhost** attribute is specified in a job definition, the scheduler tries to resolve the localhost value when it runs the job. The localhost value is resolved as follows:

- The scheduler checks the value of the LocalMachineDefinition parameter (UNIX) or the Local Machine Definition field (Windows).
- If the local machine definition setting is set to a value other than “localhost”, the scheduler searches the database for a machine definition with that name. For example, suppose that LocalMachineDefinition is set to agentmach. If an agentmach machine definition is found and all conditions are satisfied, the job runs on agentmach. If the scheduler cannot find an agentmach machine definition, or if it finds multiple agentmach machine definitions, the scheduler does not resolve localhost. All jobs defined to run on the localhost machine fail.

- If the local machine definition is not defined or is set to “localhost”, the scheduler searches the database for a machine definition corresponding to the machine where the scheduler was started (the default localhost). For example, suppose that the scheduler was started on a machine named prodserver and LocalMachineDefinition is not defined. When the job runs, the scheduler searches for a machine definition named prodserver. If the scheduler cannot find the prodserver definition, or if it finds multiple prodserver definitions, the scheduler does not resolve localhost. All jobs defined to run on the localhost machine fail.
- In a high availability failover where the shadow scheduler takes over from the primary scheduler, the localhost is resolved in the same way. To run a job on the localhost, the shadow scheduler first checks its local machine definition setting, which may be different from the setting for the primary scheduler. If the local machine definition is not defined, the localhost is resolved to the machine where the shadow scheduler was started.

Define a Machine

Before you can schedule jobs to run on a machine, you must define the machine to CA Workload Automation AE.

To define a machine

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.
2. Specify the following definition:

   ```
   insert_machine: machine_name
   node_name: address
   type: type
   ```

   machine_name

   Defines a unique name for the machine to add.

   address

   Specifies the IP address or DNS name of the machine. The default is the machine_name specified on the insert_machine statement.

   type

   Specifies the type of machine you are defining. Options are the following:

   a

   Specifies a CA Workload Automation Agent for UNIX, Linux, Windows, i5/OS, or z/OS machine. This is the default.

c

Specifies a CA AutoSys Workload Automation Connect Option machine.

l

Specifies a 4.5 real UNIX machine. You must specify a lowercase l.

L

Specifies a 4.5 real Windows machine. You must specify a capital L.

n

Specifies an r11 real Windows machine or a virtual machine that consists only of r11 real Windows machines (type n).

p

Specifies a real machine pool managed by CA Spectrum Automation Manager.

Note: In the documentation, the type "p" machine is referred to as the real machine pool.

r

Specifies an r11 real UNIX machine.

u

Specifies a CA NSM or a CA Universal Job Management Agent (CA UJMA) machine.

v

Specifies a virtual machine. The virtual machine can consist of CA Workload Automation Agent machines (type a), r11 real UNIX machines (type r), and r11 real Windows machines (type n).

3. (Virtual or CA Spectrum Automation Manager machine pool types only) Specify the following attribute:

   machine

   Specifies a real machine as a component of the virtual machine or real machine pool. The specified machine must have been defined to CA Workload Automation AE as a real machine.

4. (CA Workload Automation Agent for UNIX, Linux, Windows, i5/OS, or z/OS machine only) Specify the following attribute:

   opsys

   Specifies the operating system where the CA Workload Automation Agent is installed. Options are the following:

   aix

   Specifies a CA Workload Automation Agent for UNIX installed on an AIX computer.

   hpux

   Specifies a CA Workload Automation Agent for UNIX installed on an HP-UX computer.

   linux

   Specifies a CA Workload Automation Agent for LINUX.

   I5os

   Specifies a CA Workload Automation Agent for i5/OS.

   solaris

   Specifies a CA Workload Automation Agent for UNIX installed on a Solaris computer.

   windows

   Specifies a CA Workload Automation Agent for Windows.

   zos

   Specifies a CA Workload Automation Agent for z/OS.

5. Specify optional attributes:
   - agent_name
   - character_code
   - description
   - encryption_type
   - factor
   - heartbeat_attempts (CA Workload Automation Agent for UNIX, Linux, Windows, i5/OS, or z/OS only)
   - heartbeat_freq (CA Workload Automation Agent for UNIX, Linux, Windows, i5/OS, or z/OS only)
   - key_to_agent
   - max_load
   - opsys
   - port
6. Do *one* of the following:
   - Enter **exit** if you are using interactive mode.
   - Redirect the script to the jil command if you are using a script.

   The data is loaded into the database and the machine is defined.

Notes:

- You can only use JIL to define real and virtual machines.
- Load balancing is automatic when using real machine pools from integration with CA Spectrum Automation Manager. Without CA Spectrum Automation Manager, load balancing and queuing is possible only if real and virtual machines have been defined to CA Workload Automation AE using the machine attributes. The max_load and factor attributes, used when defining real machines, are important for load balancing and queuing.
- For more information about the insert_machine subcommand and the related machine attributes, see the *Reference Guide*.

Examples: Defining Real Machines

The following examples define real machines:

Example: Define a CA Workload Automation Agent

This example defines a machine named eagle where the agent WA_AGENT runs on the node myagenthostname and uses 49154 as its main input port.

```
insert_machine: eagle
type: a
agent_name: WA_AGENT
node_name: myagenthostname
port: 49154
max_load: 100
factor: 1.0
```

Example: Define an r11 Real Windows Machine

This example defines a Windows real machine named jaguar.

```
insert_machine: jaguar
type: n
max_load: 100
factor: 1.0
```

Example: Define an r11 Real UNIX Machine

This example defines a UNIX real machine named jaguar.

```
insert_machine: jaguar
type: r
max_load: 100
factor: 1.0
```

Examples: Defining Virtual Machines

The following examples define virtual machines:

Example: Define a Virtual Machine to Include Two Real Windows Machines

This example defines a virtual machine named giraffe to include two real r11.3 Windows machines (cheetah with a factor value of 5.0 and a max_load value of 400, and lily with a factor value of 2 and a max_load value of 15).

```
insert_machine: cheetah
type: a
opsys: windows

insert_machine: lily
type: a
opsys: windows

insert_machine: giraffe
type: v
machine: cheetah
max_load: 400
factor: 5.0
machine: lily
max_load: 15
factor: 2
```

Note: In this example, the two real machines (when specified in a job definition through the virtual machine) vary considerably in capacity from a scheduling standpoint. However, when these machines are explicitly specified by name in a job definition, the factor and max_load attributes defined here have no effect; they only have these values when used through the virtual machine.

Example: Define a Windows Virtual Machine with Subsets of r11 Real Machines

This example defines two r11 Windows real machines (lion and lotus), and a virtual machine (gorilla), which is composed of slices, or subsets, of the max_load specified for the real machines. Although the real machines were defined with specific max_load values (100 and 80), the virtual machine only makes use of the reduced loads specified in the virtual machine definition (10 and 9).

```
insert_machine: lion
type: n
max_load: 100
factor: 1

insert_machine: lotus
type: n
max_load: 80
factor: .8

insert_machine: gorilla
type: v
machine: lion
max_load: 10
machine: lotus
max_load: 9
```

Example: Define a Virtual Machine with Default Real Machines

This example defines a virtual machine (sheep), which is composed of two r11.3 UNIX real machines (warthog and camel). Because the max_load and factor attributes are not defined for the real machines, they use the default values for these attributes (a factor of 1.0 and a max_load of none, indicating unlimited load units).

```
insert_machine: warthog
opsys: linux

insert_machine: camel
opsys: solaris

insert_machine: sheep
type: v
machine: warthog
machine: camel
```

Example: Define a Virtual Machine with r11 Real Machines

This example defines two r11 UNIX real machines (lion and lotus), and a virtual machine (zebra), which is composed of the two real machines. The virtual machine is a superset of the two real machines and uses the max_load and factor attributes defined for them.

```
insert_machine: lion
type: r
max_load: 100
factor: 1

insert_machine: lotus
type: r
max_load: 80
factor: .9

insert_machine: zebra
type: v
machine: lion
machine: lotus
```

Example: Define a Real Machine Pool

This example defines a real machine pool (DCAPOOL), which is composed of three real machines (MWIN, MLIN, and MSQL). DCAPOOL is monitored by CA Spectrum Automation Manager. When you define a job to reference DCAPOOL, CA Spectrum Automation Manager is used for machine selection.

```
insert_machine: MWIN
node_name: myhost

insert_machine: MLIN
node_name: myhost1

insert_machine: MSQL
node_name: myhost2

insert_machine: DCAPOOL
type: p
machine: MWIN
machine: MLIN
machine: MSQL
```

Examples: Defining Real Machine Pools

The following example defines a real machine pool:

Example: Define a Real Machine Pool

This example defines a real machine pool (DCAPOOL), which is composed of three real machines (MWIN, MLIN, and MSQL). DCAPOOL is monitored by CA Spectrum Automation Manager. When you define a job to reference DCAPOOL, CA Spectrum Automation Manager is used for machine selection.

```
insert_machine: MWIN
node_name: myhost
insert_machine: MLIN
node_name: myhost1
insert_machine: MSQL
node_name: myhost2
insert_machine: DCAPOOL
type: p
machine: MWIN
machine: MLIN
machine: MSQL
```

Delete a Real Machine

To delete a real machine definition, specify the following subcommand in the JIL script:

```
delete_machine: name_of_real_machine_to_delete
```

If the real machine is referenced by virtual machines or real machine pools, you must remove all references from all virtual machines and real machine pools to delete the machine. You must either delete the real machine from each of the virtual machines or pools manually prior to deleting the real machine or explicitly request JIL to do this for you. If you request JIL remove the references, you may end up with virtual machines or pools that contain no machines resulting in jobs that will not run. Use the following syntax to request JIL delete the real machine and all references to it from virtual machines and pools:

```
delete_machine: name_of_real_machine_to_delete
remove_references: y
```

You can force a machine to be deleted whether it is currently being used or not. Use the following syntax:

```
delete_machine: name_of_real_machine_to_delete
force: y
```

Note: For more information about deleting real machines and the related attributes, see the *Reference* Guide.

Example: Delete a Real Machine Not Referenced in Virtual Machines or Real Machine Pools

This example deletes the real machine definition for the computer named jaguar:

```
delete_machine: jaguar
```

Example: Delete a Real Machine Currently Referenced via a Virtual Machine:

This example explicitly deletes the real machine hyena reference from the virtual machine carnivores followed by the real machine definition itself.

```
delete_machine: carnivores
machine: hyena
delete_machine: hyena
```

Example: Delete a Real Machine and All Virtual Machines References Implictly:

This example deletes the real machine panther and implicitly deletes all references to it that may be in any virtual machines or real machine pools.

```
delete_machine: panther
remove_references: y
```

Delete a Virtual Machine

To delete a virtual machine, specify the delete_machine: *machine_name* subcommand without the machine attribute in the JIL script. When you delete a virtual machine, the definitions for its component real machines are not deleted.

You can delete all real machine references in a virtual machine until there is only one reference remaining. You cannot delete the last reference. To delete all real machine references in a virtual machine, you must also delete the virtual machine itself.

Note: For more information about deleting virtual machines and the related attributes, see the *Reference Guide*.

Example: Delete a Virtual Machine

This example deletes the virtual machine definition named gorilla:

```
delete_machine: gorilla
```

Delete a Real Machine Pool

To delete a real machine pool, specify the delete_machine: *machine_name* subcommand without the machine attribute in the JIL script. When you delete a real machine pool, the definitions for its component real machines are not deleted.

You can delete all real machine references in a real machine pool until there is only one reference remaining. You cannot delete the last reference. To delete all real machine references in a real machine pool, you must also delete the real machine pool itself.

Note: For more information about deleting real machine pools and the related attributes, see the *Reference Guide*.

Example: Delete a Real Machine Pool

This example deletes the real machine pool definition named gorilla:

```
delete_machine: gorilla
```

Delete a Real Machine from a Virtual Machine or Real Machine Pool

To delete a virtual machine or real machine pool reference to a real machine, specify the following subcommand in the JIL script:

```
delete_machine: virtual_machine_name
machine: real_machine_name_referenced
```

Example: Delete a Real Machine from a Virtual Machine

This example deletes the real machine named camel from the virtual machine named sheep. The machine definitions for sheep and camel are *not* deleted from the database.

```
delete_machine: sheep
machine: camel
```

Specifying Machine Load (max_load)

You can use the max_load attribute to define the maximum load (in load units) that a machine can reasonably handle. The max_load attribute is valid in a real machine definition or component machines defined to virtual machines.

Load units are arbitrary values, the range of which is user-defined. You can use any weighting scheme you prefer. For example, a load unit with a range of 10 to 100 would specify that machines with limited processing power are expected to carry a load of only 10, while machines with ample processing power can carry a load of 100. There is no direct relationship between the load unit value and any of the machine's physical resources. Therefore, we recommend that you use conventions that are meaningful to you. You cannot use zero (0) or negative numbers as load units.

The max_load attribute is primarily used to limit the load on a machine. As long as a job's load will not exceed a machine's maximum load, the max_load attribute does not influence which machine a job runs on.

If you do not define the max_load attribute in a machine definition, CA Workload Automation AE does not limit the load on the machine.

Example: Set the Maximum Load for a Real Machine

Suppose that the range of possible load values is 1 to 100. This example sets the maximum load for a relatively low-performance real machine.

```
max_load: 20
```

Specifying Job Load (job_load)

For load balancing to work, you must assign a job_load value to every job that impacts the load on a machine. The job_load attribute in a job definition defines the relative amount of processing power the job consumes (the relative load the job places on a machine).

Load units are arbitrary values, and the range is user-defined. You can use any weighting scheme you prefer. You can use the max_load attribute to assign a real machine a maximum job load. Then, you can use the job_load attribute in the job definition to assign the job a load value that indicates the relative amount of the machine's load that the job should consume. These attributes let you control machine loading and prevent a machine from being overloaded.

Example: Define the Relative Processing Load for a Job

Suppose that the range of possible load values is 1 to 100. This example sets the load for a job that typically uses 10% of the CPU.

```
job_load: 10
```

Specifying Queuing Priority (priority)

When a job is ready to run on a designated machine but the current load on that machine is too large to accept the new job's load, CA Workload Automation AE queues the job for that machine so it runs when sufficient resources are available.

For job queuing to take place, you must define the priority attribute in the job definition. The queue priority establishes the relative priority of all jobs queued for a given machine. The lower number indicates a higher priority. If you do not set the priority attribute or the priority is set to 0, the job runs immediately on a machine and is not put in the queue. The job ignores any other job or machine load settings defined.

Example: Set the Job to Run with Highest Priority

This example sets the job to run with the highest priority without overriding the machine load control mechanism.

```
priority: 1
```

Example: Set the Job to Run in the Background

This example sets the job to run in the background when the machine load is low.

```
priority: 100
```

Specifying Relative Processing Power (factor)

You can use the factor attribute to check the relative available CPU cycles. The factor attribute defines a factor to multiply by a machine's available CPU cycles. The factor value is used to weigh available cycles on one machine against those of another machine. When CA Workload Automation AE checks the available cycles on each machine, it multiplies the percent of free CPU cycles by the factor value to check which machine has more relative processing power available.

The factor attribute is valid in a real machine definition or component machines defined to virtual machine.

Factor units are arbitrary values, and the range is user-defined. The value consists of a real number that can contain a decimal. Therefore, the factor value is typically a number between 0.0 and 1.0. If you do not define the factor attribute in a machine definition, its value defaults to 1.0.

Example: Set the Factor for a Low-Performance Real Machine

This example sets the factor for a low-performance real machine, on a scale of 0.0 to 1.0.

```
factor: 0.1
```

Example: Set the Factor for a High-Performance Real Machine

This example sets the factor for a high-performance real machine, on a scale of 0.0 to 1.0.

```
factor: 1.0
```

Machine Status

Real machines have a run-time status attribute designed to reflect the machine’s availability. The machine status lets the scheduler run more efficiently by not wasting time trying to contact machines that are out of service. If a job is scheduled for a machine that is offline, it is set to PEND_MACH status until the machine comes back online. In the case of a virtual machine, offline machines are not considered as possible candidates for running a job.

A machine can have one of following statuses:

Online

Indicates that the machine is available and accepting jobs to run.

Offline

Indicates that the machine has been manually removed from service and will not accept jobs to run.

Missing

Indicates that the scheduler has verified that the machine is not responding and has automatically removed it from service. The machine will not accept jobs to run.

Unqualified

Indicates that the scheduler is attempting to qualify the status of an agent before switching the machine from an online to missing status. The machine will not accept jobs to run.

Empty

Indicates that a virtual machine or real machine pool does not contain any component machines. Jobs scheduled to machines in this status will not run.

Take a Machine Offline Manually

To manually take a machine offline (for example, during hardware service), use the sendevent command to send a MACH_OFFLINE event.

When you send a MACH_OFFLINE event, jobs that are currently running run to completion even though the machine's status is offline. You can use the autorep command to monitor running jobs.

If you shut a machine down for servicing, you may want to let the running jobs complete before continuing. With the machine offline, you can service the machine while the scheduler continues running. All jobs that are scheduled to start on the offline machine are put in PEND_MACH status until the machine returns to service.

Note: For more information, see the *Reference Guide*.

Example: Manually Take a Machine Offline

This example takes the machine cheetah offline:

```
sendevent -E MACH_OFFLINE -n cheetah
```

The scheduler log displays a message similar to the following when the machine is offline:

```
[11/28/2005 15:38:21]     CAUAJM_I_40245 EVENT: MACH_OFFLINE     MACHINE: cheetah
```

Put a Machine Online Manually

To manually put a machine online, use the sendevent command to send a MACH_ONLINE event.

When you send a MACH_ONLINE event for a machine, jobs with a status of PEND_MACH on that machine are automatically started.

Note: For more information, see the *Reference Guide*.

Example: Manually Put a Machine Online

This example returns the machine cheetah to online status:

```
sendevent –E MACH_ONLINE –n cheetah
```

The scheduler log displays a message, similar to the following, when the machine is online:

```
[11/28/2005 15:38:21]     CAUAJM_I_40245 EVENT: MACH_ONLINE     MACHINE: cheetah
```

How Status Changes Automatically

When the scheduler verifies that a real machine is not reachable, it uses the following process to manage machine and job status:

- If the scheduler fails to contact a machine's agent, the scheduler marks the machine as unqualified and logs a message similar to the following:

  ```
  [11/28/2005 16:01:46]    CAUAJM_W_40290 Machine cheetah is in question. Placing machine in the unqualified state.
  ```

- The scheduler puts all jobs scheduled to start on the unqualified machine in PEND_MACH status.
- The scheduler attempts to qualify the status of that machine by pinging the agent every 10 seconds
- If the agent responds, the scheduler sends a MACH_ONLINE event and the machine returns to service.
- When the machine returns to service, the scheduler starts all jobs in PEND_MACH status for that machine.
- If the agent fails to respond after three attempts, the scheduler marks the machine as missing, issues a MACHINE_UNAVAILABLE alarm, and logs a message similar to the following:

  ```
  [11/28/2005 16:01:46]   CAUAJM_I_40253 Machine cheetah is not responding.  Taking offline.
  ```

- The scheduler puts all jobs scheduled to start on the missing machine in PEND_MACH status.
- If the machine definition is updated, the scheduler marks the machine as unqualified, logs the following message, and pings the agent until the machine returns to service or is marked missing:

  ```
  [11/28/2005 16:01:46]   CAUAJM_W_40291 Machine cheetah has been updated. Placing machine in the unqualified state.
  ```

- Otherwise, the scheduler pings the missing machine's agent every 60 seconds to check its availability.
- If the agent responds, the scheduler sends a MACH_ONLINE event and the machine returns to service.
- When the machine returns to service, the scheduler starts all jobs in PEND_MACH status for that machine.

  Note: If you understand the cause of a missing machine and intervene to correct it, you can use the sendevent command to send a MACH_ONLINE event to bring the machine back online instead of waiting for the scheduler to do so.

How Status Affects Jobs on Virtual Machines

If a job is defined to run on a virtual machine or a list of machines and one of those machines is offline, the job will run on another available machine with which it is associated.

If, however, all machines in the virtual list are offline, the scheduler puts the job in PEND_MACH status. If any of the machines with which the job is associated comes back online, the scheduler removes the job from PEND_MACH status and runs it on the online machine, subject to the queuing criteria.

Load Balancing

Load balancing can be implemented to use inherent features of CA Workload Automation AE or integrated with CA Spectrum Automation Manager. The usage of real machine pools provides automatic load balancing via CA Spectrum Automation Manager.

When not using CA Spectrum Automation Manager for load balancing, you can implement load balancing (where the workload is spread across multiple machines based on each machine's capabilities) by using the machine attribute to specify a virtual machine or multiple real machines in a job definition. This is also an easy way to help ensure reliable job processing. For example, the scheduler can use load balancing to check which of the machines in a job definition is best suited to run the job, and automatically start it on that machine.

The advantages of building a virtual machine are as follows:

- Its definition can be changed and the new construct is immediately applied globally.
- The max_load and factor values can vary between machines.

Alternatively, you can specify a list of real machines in the job's machine attribute. The system configuration includes machines of varying processing power. CA Workload Automation AE uses one of various load balancing methods to choose a real machine. If you specify the cpu_mon or vmstat load balancing methods in the configuration file, CA Workload Automation AE chooses which machine to run on based on the available processing power obtained from the agent. If you specify the job_load load balancing method in the configuration file, CA Workload Automation AE chooses which machine to run on based on the max_load and factor attributes for each real machine in conjunction with the job definition’s priority and job_load attributes. If you specify the UNIX-only rstatd load balancing method, in the configuration file, CA Workload Automation AE chooses which machine to run on based on the information obtained from the remote UNIX computer’s remote kernel statistics daemon.

In either case, CA Workload Automation AE uses the following process to verify the available relative processing cycles for each machine:

1. CA Workload Automation AE calculates the number of load units available on each real machine in the specified virtual machine. To do this, CA Workload Automation AE uses the load balancing method specified in the configuration file.

   Notes:

   - For the CA Workload Automation Agent on UNIX, Linux, Windows, or i5/OS (machine type: a), CA Workload Automation AE uses cpu_mon, rstatd, or the job_load method. If the machine method specified in the configuration file is set to the cpu_mon or vmstat methods, the scheduler runs a CPU Monitoring (OMCPU) job to determine the available CPU cycles. This is the default. For the CA Workload Automation Agent on UNIX and Linux only (opsys: aix, hpux, linux, or solaris), CA Workload Automation AE supports the rstatd method.
   - For the legacy agent on UNIX, CA Workload Automation AE only uses vmstat, rstatd, or the job_load method.
   - For the agent on Windows, CA Workload Automation AE only uses vmstat or the job_load method.

2. CA Workload Automation AE performs the following calculation:

   ```
   Machine Usage = Available Load Units * Factor value
   ```

3. CA Workload Automation AE chooses the machine with the most relative load units available, based on the calculation in Step 2.

Notes:

- If a real machine in the virtual machine is not online, the scheduler does not attempt to contact it and it is not considered in the load balancing algorithm.
- If the machines have equal max_load and factor values, it is equivalent to defining a job and specifying the following in the machine field:

  ```
  machine: cheetah, camel
  ```

- If the factor attribute is not specified for a machine, CA Workload Automation AE assumes the default factor value for each machine (1.0).
- On UNIX, the load balancing method is specified using the MachineMethod parameter in the CA Workload Automation AE configuration file. On Windows, the method is specified using the Machine Method field on the Scheduler window of CA Workload Automation AE Administrator. For more information, see the *Administration Guide* or *Online Help* for CA Workload Automation AE Administrator.
- The cpu_mon machine method does not apply to z/OS machines (CA Workload Automation Agent on z/OS) because the OMCPU job is not supported on z/OS.
- If the load balancing request is sent to a legacy agent, CA Workload Automation AE uses the vmstat method to obtain the available CPU cycles.

Example: Load Balancing With a Virtual Machine

This example defines a virtual machine (marmot) with three real machines (cheetah, hippogriff, and camel):

```
insert_machine: marmot
machine: cheetah
factor: 1
machine: hippogriff
factor: .8
machine: camel
factor: .3
```

To start a job on this virtual machine, specify marmot in the job's machine attribute. The scheduler performs the necessary calculations to verify on which machine to run the job, and reflects these calculations in its output log. The output is similar to the following:

```
EVENT: STARTJOB JOB: test_mach
[11/22/2005 10:16:53]     CAUAJM_I_40245 EVENT: STARTJOB        JOB: tvm
[11/22/2005 10:16:54]     CAUAJM_I_10208   Checking Machine usages:
[11/22/2005 10:16:59]     <cheetah=78>
[11/22/2005 10:17:02]     <hippogriff=80*[.80]=64>
[11/22/2005 10:17:07]     <camel=20*[.30]=6>
[11/22/2005 10:17:11]     CAUAJM_I_40245 EVENT: CHANGE_STATUS    STATUS: STARTING
JOB: tvm
[11/22/2005 10:17:11]     CAUAJM_I_10082 [cheetah connected for tvm]
```

Note that even though the machine usage on cheetah was less than that of machine hippogriff, machine cheetah was picked because of the result of the factor calculation (machine cheetah had 78% of its processing power available, while machine hippogriff only had 64% available). Thus, the factors weigh each machine to account for variations in processing power.

Load Balancing Using Virtual Resource Dependencies

Load balancing can also be performed using virtual resources combined with virtual machines or machine lists for basic throttling and serialization. If you assign a virtual machine or list of real machines to a job along with virtual resource dependencies, the jobs can be dispatched to the various machines based on resource availability.

Example: Load Balancing Using Machine Virtual Resources

Suppose that you have three machines (sloth, tiger, and leopard) capable of running several applications but they vary in capacity or physical resources, for example CPU speed, memory, or utilization. You also have various jobs that use different runtime resources. Jobs with low resource usage can run anywhere. Jobs that use more resources are limited to where they can run. The number of concurrent jobs and their requirements must be controlled to avoid overburdening any machine.

- Define the real machine definitions for sloth, tiger, and leopard. Then define a virtual machine, domain, that references all the machines where the jobs should be allowed to run.

  ```
  insert_machine: sloth
  insert_machine: tiger
  insert_machine: leopard
  insert_machine: domain
  type: v
  machine: sloth
  machine: tiger
  machine: leopard
  ```

- Define the maximum amount of virtual resources available to each machine. Remember, they are virtual resources. They do not really exist. The virtual resource amounts are approximations based on estimated capabilities of the machines. In this example, the sloth machine has the fewest capabilities while the leopard machine has the most.

  ```
  insert_resource: job_weight
  res_type: r
  machine: sloth
  amount: 2
  insert_resource: job_weight
  res_type: r
  machine: tiger
  amount: 10
  insert_resource: job_weight
  res_type: r
  machine: leopard
  amount: 30
  ```

- Define jobs to run on one of the real machines referenced by the virtual machine domain. Identify the virtual resource units each job consumes. Similar to the resource definition, these values are approximations based on perceptions or expectations of what the job consumes while running. The quantity required for the job determines where it can run and the mix of jobs that can run concurrently.

```
insert_job: quick_job
machine: domain
command: efficient_report
resources: (job_weight,quantity=1,free=y)
insert_job: heavy_job
machine: domain
command: analytical_report
resources: (job_weight,quantity=5,free=y)
insert_job: beastly_job
machine: domain
command: quarterly_update
resources: (job_weight,quantity=10,free=y)
```

Based on the above definitions, job quick_job could run on any of the machines defined to the virtual machine named domain because all machines have at least one unit of the job_weight virtual resource defined to it. The jobs heavy_job and beastly_job can only be scheduled to real machines tiger and leopard. The two jobs cannot be scheduled concurrently to the tiger machine as that would exceed the virtual resources defined to it. If the job heavy_job is already running on the tiger machine when job beastly_job is being scheduled, the job beastly_job would be scheduled to run on the leopard real machine.

Load Balancing Using Virtual and Real Resource Dependencies

You can also implement load balancing using virtual and real resources as dependencies to a job. Virtual and real resource dependencies can be defined to both virtual machines and real machine pools.

If you assign a virtual machine or real machine pool to a job with either virtual and/or real resource dependencies, the job runs on the machine that satisfies the resource dependencies. If the job has real resource dependencies and two or more machines satisfy the specified metrics, CA Spectrum Automation Manager returns the best machine based on the lowest overall utilization. If the job has only virtual resource dependencies and two or more machines satisfy the specified metrics, the job runs on the first machine that satisfies the virtual resource requirements.

Note: If CA Workload Automation AE is not integrated with CA Spectrum Automation Manager, real resource dependencies are ignored and the job is submitted on the first machine that satisfies the virtual resource requirements. If the job does not have virtual resources, it runs on the machine as determined by load balancing using the max_load and factor values.

Example: Load Balancing Using Virtual and Real Resource Dependencies

Suppose that you want to define a job that gets submitted on a machine that satisfies the real and virtual resource dependencies, you can do the following:

- Define a renewable resource ren_glb1 at the global level:

  ```
  insert_resource: ren_glb1
  res_type: R
  amount: 10
  ```

- Define a real machine pool DCAPOOL to include three real machines (MWIN, MLIN, and MSOL) that are discovered and monitored by CA Spectrum Automation Manager for real time load balancing:

  ```
  insert_machine: DCAPOOL
  type: p
  machine: MWIN
  machine: MLIN
  machine: MSOL
  ```

- Define a job job_load with real and virtual resource dependencies:

```
insert_job: job_load
job_type: CMD
command: sleep 1
machine: DCAPOOL
owner: autosys
date_condition: 0
alarm_if_fail: 1
resources: (ren_glb1, quantity=2, free=y) and (MEM_INUSE_PCT, VALUE=30,
VALUEOP=LTE)
```

- Generate a report for the job (job_load) to view whether the real and virtual resource dependencies are satisfied on the MWIN, MLIN, and MSOL machines:

```
job_depends -J job_load -r
```

The report might resemble the following:

```
Job Name            Machine
--------            ----------
job_load            MLIN
    Virtual Resources
    -----------------
Resource            Type    Amount  Satisfied?
--------            ----    ------  ----------
ren_glb1            R       2       YES
    Real Resources
    -----------------
Resource                                    Satisfied?
-----------                                 -----------
MEM_INUSE_PCT, VALUE=30, VALUEOP=LTE        NO
----------------------------------------------------------------------------
Job Name            Machine
--------            ----------
job_load            MSOL
   Virtual Resources
    -----------------
Resource            Type    Amount  Satisfied?
--------            ----    ------  ----------
ren_glb1            R       2       YES
    Real Resources
    -----------------
Resource                                    Satisfied?
-----------                                 -----------
MEM_INUSE_PCT, VALUE=30, VALUEOP=LTE        YES
----------------------------------------------------------------------------
Job Name            Machine
--------            ----------
job_load            MWIN
```

```
   Virtual Resources
    -----------------
Resource           Type    Amount  Satisfied?
--------           ----    ------  ----------
ren_glb1           R       2       YES
    Real Resources
    -----------------
Resource                                   Satisfied?
-----------                                -----------
MEM_INUSE_PCT, VALUE=30, VALUEOP=LTE       YES
```

The report displays that the virtual resource (ren_gbl1) is satisfied on all the machines. However, the real resource MEM_INUSE_PCT is satisfied only on MWIN and MSOL machines. When you start the job (job_load), CA Spectrum Automation Manager decides the best machine with the least overall utilization between the MWIN and MSOL machines.

- Start the job (job_load):

  ```
  sendevent –E START_JOB –J job_load
  ```

- Generate a detailed report for the job (job_load) to view the machine on which the job runs:

  ```
  autorep -J job_load -d
  ```

  The resulting report might resemble the following:

```
Job Name                    Last Start           Last End             ST Run/Ntry Pri/Xit

___________________________ ____________________ ____________________ __ ________ _______
job_load            10/07/2010 15:06:21  10/07/2010 15:06:23  SU 689/1    0

   Status/[Event]  Time                 Ntry ES  ProcessTime           Machine
   --------------  --------------------- --   --  --------------------- ----------------
   STARTING        10/07/2010 15:06:21    1  PD  10/07/2010 15:06:22   MSOL
   RUNNING         10/07/2010 15:06:22    1  PD  10/07/2010 15:06:22   MSOL
     <Executing at WA_AGENT>
   SUCCESS         10/07/2010 15:06:23    1  PD  10/07/2010 15:06:23   MSOL
```

The report displays that CA Spectrum Automation Manager decided MSOL as the best machine with the least overall utilization.

Load Balancing Using Real Resource Pools

If you assign the real machine pool to a job without any real resource dependencies, CA Spectrum Automation Manager monitors these machines and decides the best machine with the least overall utilization for job submission.

Note: This does not apply to virtual machines although you may create a virtual machine that is composed of real machines that are monitored by CA Spectrum Automation Manager.

Example: Load Balancing Using Real Machine Pools

Suppose that you want to define a job that gets submitted on a machine with least overall utilization, you can do the following:

- Define a real machine pool DCAPOOL to include three real machines (MWIN, MLIN, and MSOL) that are discovered and monitored by CA Spectrum Automation Manager for real time load balancing:

  ```
  insert_machine: DCAPOOL
  type: p
  machine: MWIN
  machine: MLIN
  machine: MSOL
  ```

- Define a job job_load and assign the real machine pool DCAPOOL to it:

  ```
  insert_job: job_load
  machine: DCAPOOL
  command: sleep 1
  owner: autosys
  ```

  CA Spectrum Automation Manager monitors the three machines (MWIN, MLIN, and MSOL) and decides the best machine with the least overall utilization for job submission. For example, if the overall utilization of MWIN is 68%, MLIN is 50%, and MSOL is 56%, CA Spectrum Automation Manager selects MLIN machine for job submission.

- Start the job (job_load):

  ```
  sendevent –E START_JOB –J job_load
  ```

- Generate a detailed report for the job (job_load) to view the machine on which the job runs:

  ```
  autorep -J job_load -d
  ```

  The resulting report might resemble the following:

```
Job Name                    Last Start           Last End             ST Run/Ntry Pri/Xit
___________________________ ____________________ ____________________ __ ________ _______
job_load            10/07/2010 14:35:42  10/07/2010 14:35:43  SU 687/1    0

   Status/[Event]  Time                 Ntry ES  ProcessTime           Machine
   --------------  --------------------- --  --  --------------------- ----------------
   STARTING        10/07/2010 14:35:42    1  PD  10/07/2010 14:35:42   MLIN
   RUNNING         10/07/2010 14:35:42    1  PD  10/07/2010 14:35:42   MLIN
     <Executing at WA_AGENT>
   SUCCESS         10/07/2010 14:35:43    1  PD  10/07/2010 14:35:43   MLIN
```

Forcing a Job to Start

If you use the sendevent command to send a FORCE_STARTJOB event to a job, CA Workload Automation AE immediately starts the job on the machine specified in the job definition, regardless of the current load on the machine or the job_load value set for the job. If the job was defined to run on a virtual machine or a list of real machines, CA Workload Automation AE checks which machine has the most processing power available and runs the job on that machine, even if the job_load value set for the job exceeds the max_load value set for the machine.

Notes:

- If you send a FORCE_STARTJOB event to a job that has a status of ON_ICE or ON_HOLD, the job's status does not revert to its previous status when it completes.
- If you send a FORCE_STARTJOB event to a job that has a status of RESWAIT, the FORCE_STARTJOB is ignored and the job remains in the RESWAIT status. You can remove or alter the resource requirements of the job so the job is no longer in RESWAIT and can be started.

Example: Force a Job to Start

This example describes the effects of forcing a job to start. Assume you scheduled Job1 to run every Monday at 3:00 A.M. On Sunday, you sent a JOB_ON_HOLD event to put the job in ON_HOLD status, so that the job does not run as scheduled on Monday. If you send a FORCE_STARTJOB event to Job1 on Wednesday at 2:00 P.M., Job1 runs to completion (either success or failure), and then runs again as scheduled on Monday at 3:00 A.M. Note that the job did not revert to the ON_HOLD status after you forced it to start on Wednesday.

Queuing Jobs

Queuing is a mechanism used in CA Workload Automation AE to check the run order of jobs that cannot run immediately. There is no actual physical queue. Instead, CA Workload Automation AE uses queuing policies, which are based on the use and subsequent interaction of the job_load and priority attributes in a job definition and the max_load and factor attributes in a machine definition. You can also use the sendevent command to send a CHANGE_PRIORITY event to change the priority of a job that is already queued.

Note: The constructs of job loads (specified in the job_load attribute) and machine loads (specified in the max_load attribute) are user-defined conventions that CA Workload Automation AE enforces. If you do not indicate what the load of a job is, it does not figure in the product's queuing calculations. This is different from the load balancing feature, which inspects the CPU to verify its usage.

The following sections discuss queuing jobs and give examples of how to use load balancing and queuing to optimize job processing in your environment.

How CA Workload Automation AE Queues Jobs

For queuing to be most effective, you must set the priority attribute for all jobs. By default, the priority attribute is set to 0, indicating that the job should not be queued and should run immediately. When you let the priority attribute default for a job, it runs even if its job load would push the machine over its load limit. However, even when jobs have a priority of 0, CA Workload Automation AE tracks job loads on each machine so that jobs with non-zero priorities can be queued.

Note: If the job has resource dependencies, CA Workload Automation AE does not use the following process to limit the job load on machines and to queue jobs for processing. Instead, the resource manager (CA Workload Automation AE) is used to select the best machine to run the job.

CA Workload Automation AE uses the following process to limit the job load on machines and to queue jobs for processing:

- If you set a job_load value for a job and you assigned a max_load for every real machine comprising a virtual machine, CA Workload Automation AE checks if each machine has sufficient available load units before running the job.

  When more than one job is queued, the priority value is considered first when deciding which job to run next. If there are insufficient load units available to run the highest priority job, no other priority jobs are considered subsequently.

- If each real machine has sufficient load units, CA Workload Automation AE employs the load balancing and factor algorithms to verify on which machine the job should start.
- If only one of the machines has sufficient load units, the job runs on that machine.
- If none of the machines has sufficient load units, CA Workload Automation AE puts the job in QUE_WAIT status for all the machines. The job stays in QUE_WAIT status until one of the machines has sufficient load units available.

Note: If a job is in QUE_WAIT status and you want it to run immediately, do not force the job to start. Instead, use the sendevent command to send a CHANGE_PRIORITY event that changes the job's priority to 0.

Example: Job Queuing

This example shows a simple job queuing scenario that uses a previously defined machine named lion with a max_load of 100:

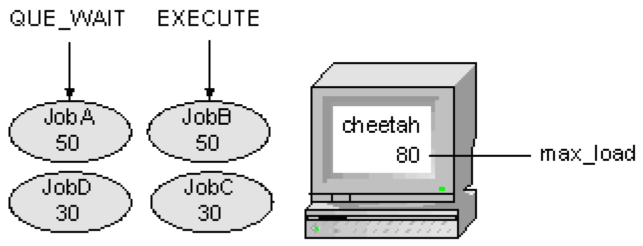
```
insert_job: jobA
machine: lion
job_load: 80
priority: 1

insert_job: jobB
machine: lion
job_load: 90
priority: 1
```

In this example, if JobA was running when JobB started, CA Workload Automation AE would put JobB in QUE_WAIT status until JobA completes, at which point JobB can run.

Example: Job Queuing and Load Balancing

This example shows a situation in which a machine has 80 load units and multiple jobs are waiting to start. In this example, JobB and JobC are executing, while JobA and JobD are queued (in the QUE_WAIT state) and waiting for available load units. The numbers in the following illustration indicate the job_load assigned to each job, and the max_load set for the machine.



The following JIL statements define the machine and the jobs in this example:

```
insert_machine: cheetah
max_load: 80

insert_job: JobA
machine: cheetah
job_load: 50
priority: 70

insert_job: JobB
machine: cheetah
job_load: 50
priority: 50

insert_job: JobC
machine: cheetah
job_load: 30
priority: 60

insert_job: JobD
machine: cheetah
job_load: 30
priority: 80
```

In this example, JobB and JobC are already running because their starting conditions were satisfied first. After JobB or JobC completes, JobA is considered to start before JobD because JobA has a higher priority.

How soon JobA starts is determined by a combination of its priority and job_load attributes, and the max_load machine attribute. The result differs based on whether JobB or JobC finishes first, as follows:

- If JobB finishes first, 50 load units become available, so JobA runs. After JobA or JobB complete, sufficient load units become available, so JobD runs.
- If JobC finishes first, only 30 load units become available, so both JobA and JobD remain queued until JobB completes.
- After JobB completes, a total of 80 load units become available, so both JobA and JobD are eligible to run. Because JobA has a higher priority, it runs first. JobD runs shortly after.

Using a Virtual Machine as a Subset of a Real Machine

One variety of virtual machine can be considered a subset of a real machine. Typically, you would use this type of virtual machine to construct an individual queue on a given machine. One use for this construct might be to limit the number of jobs of a certain type that run on a machine at any given time.

Example: Define a Virtual Machine as a Subset of a Real Machine

This example shows how to define a virtual machine that functions as a subset of a real machine, thereby acting as a queue.

In this example, cheetah is a real machine with a max_load value of 80. If you create three different print jobs, but you want only one job to run on a machine at a time, you can use a combination of the max_load attribute for a virtual machine and the job_load attributes for the jobs themselves to control how the jobs run.

To implement this scenario, you would first create the virtual machine named cheetah_printQ as follows:

```
insert_machine: cheetah_printQ
machine: cheetah max_load: 15
```

Next, you would define the three print jobs as follows:

```
insert_job: Print1
machine: cheetah_printQ
job_load: 15
priority: 1

insert_job: Print2
machine: cheetah_printQ
job_load: 15
priority: 1

insert_job: Print3
machine: cheetah_printQ
job_load: 15
priority: 2
```

Although the real machine cheetah has a max_load value of 80, meaning that all three jobs (with their job_load values of 15) could run on it simultaneously, the virtual machine cheetah_printQ effectively resets the real machine's max_load to 15. Because each job is defined to run on cheetah_printQ, not cheetah, only one of the jobs can run at a time because each job requires all of the load units available on the specified machine.

Note: The load units associated with a virtual machine have no interaction with the load units for the real machine. This example implies that the virtual load of 15 does not subtract from the load units of 80 for the real machine. Load units are simply a convention that lets the user restrict concurrent jobs running on any one machine.

Using a Virtual Machine to Combine Subsets of Real Machines

You can also define virtual machines to combine subsets (or slices) of real machines into one virtual machine. You might do this, for example, if there are two machines that are print servers and you want only one print job to run at a time on each.

Example: Define a Virtual Machine to Combine Subsets of Real Machines

This example defines a virtual machine (printQ) that uses subsets of the loads available on two real machines to control where jobs run.

To implement this, you would create the virtual machine named printQ, and specify two real machines (cheetah and camel), as shown in the following JIL statements:

```
insert_machine: printQ
type: v
machine: cheetah
max_load: 15
machine: camel
max_load: 15
```

When a job is ready to start on printQ, CA Workload Automation AE checks if the component real machine (cheetah or camel) has enough load units available to run the job.

- If neither machine has enough available load units, the product puts the job in QUE_WAIT status and starts it when there are enough load units.
- If only one machine has enough available load units, the product starts the job on that machine.
- If both machines have enough available load units, the product checks the usage on each, and starts the job on the machine with the most available CPU resources.

User-Defined Load Balancing

As an alternative to using the load balancing methods that CA Workload Automation AE provides, you can write your own programs or batch files to check which machine to use at run time. If you specify the name of a program or batch file as the value of the machine attribute in the job definition, the scheduler runs the batch file at job run time, and substitutes its output for the machine name.

If the machine returned by the script is offline, the product puts the job in PEND_MACH status for that machine. When the missing machine returns to service, the pending job runs on it regardless of whether the script would return a different machine name at that point in time. Because a machine must be defined for the scheduler to run a job on it, you must have previously defined the machine returned by the script to CA Workload Automation AE.

Example: User-Defined Load Balancing

This example shows how you would specify a user-defined program or batch file in place of a real or virtual machine for processing a job.

For example, you might supply the following:

```
insert_job: run_free
machine: '/usr/local/bin/pick_free_mach'
command: $HOME/DEL_STUFF
```

At run time, the script /usr/local/bin/pick_free_mach runs on the scheduler machine. The standard output is substituted for the name of the machine, and the job runs on that machine.

Important! The escape character in the machine value above is the back-tic character (`), not an apostrophe ('). You must escape a program or batch file used as the machine attribute value with back-tic characters as shown for the scheduler to recognize that the machine value specifies a script. The apostrophe and quotation mark characters do not work in this case.

Chapter 4: Working with Jobs

This section contains the following topics:

Jobs

All activity controlled by CA Workload Automation AE is based on jobs. A *job* is any single command or executable, UNIX shell script, or Windows batch file. Other objects, such as monitors, reports, and the Job Status Console, track job progress. A job is the foundation for the entire operations cycle.

You define jobs to CA Workload Automation AE by creating job definitions. Each job definition contains attributes that specify the job's properties and behavior. For example, you can specify conditions that determine when and where a job runs.

You can define jobs using the following methods:

JIL

A scripting language that lets you define and modify assets such as jobs, global variables, machines, user-defined job types, external instances, and blobs.

CA WCC

A tool that lets you interactively set the attributes that describe when, where, and how a job should run. The fields in the CA WCC GUI correspond to the JIL subcommands and attributes. In addition, from the CA WCC GUI, you can define calendars, global variables, and monitor and manage jobs. CA WCC is supplied with CA Workload Automation AE.

Both methods set the same attributes and the job definition is always stored in the database. You can also update and delete existing jobs.

Notes:

- The scheduler must be running before you start any processes, so you should start it before performing the tasks described in this chapter.
- For information about using the CA WCC GUI to define jobs, see the CA WCC documentation.
- Before you update or delete an existing job, ensure that the job is not running. To help avoid losing job definitions in the event of a system failure, we recommend that you back up your job definitions periodically.

More information:

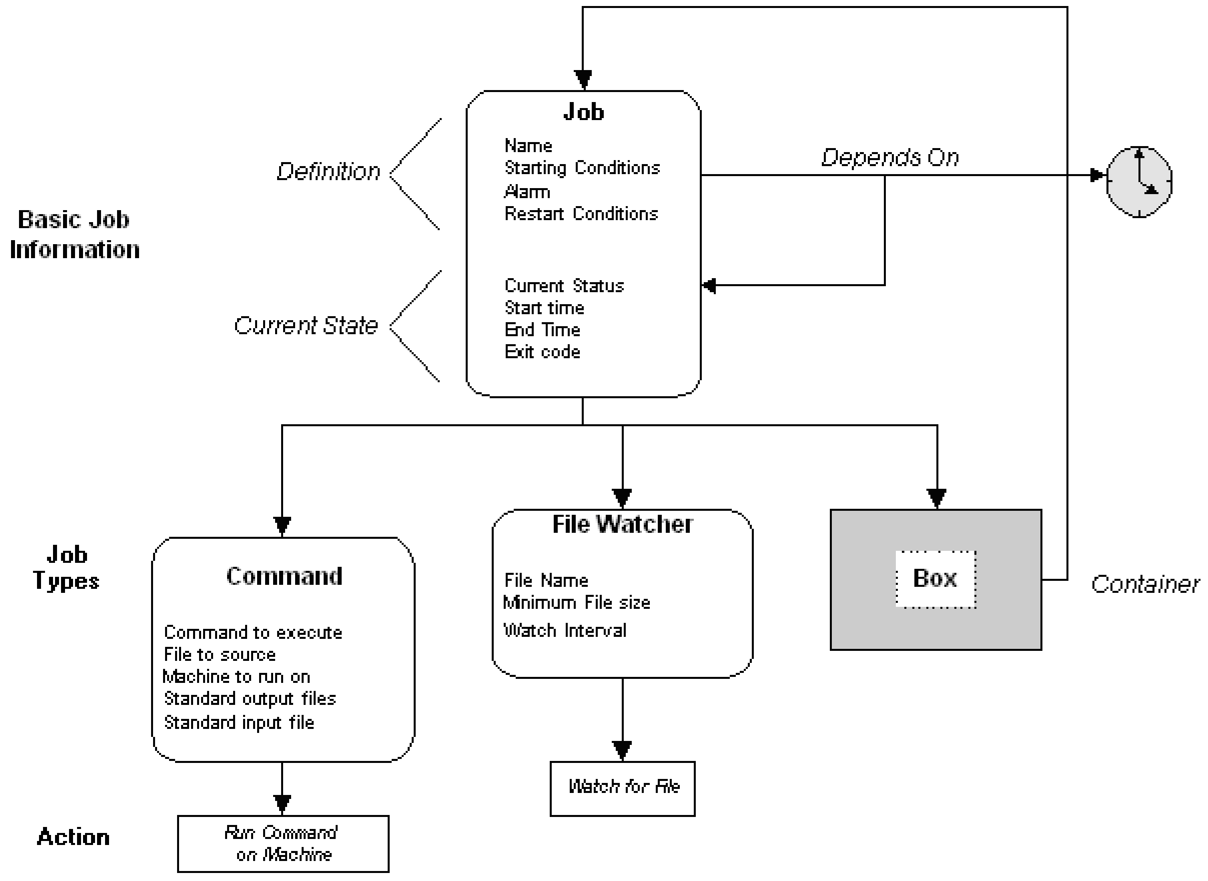
The jil Command and JIL (Job Information Language) (see page 27)

Job Types

When you create a job definition, you must specify the job type. Job types define the type of work to be scheduled. For example, you can create a CMD job to run a Windows command, an FTP job to download a file from a server, or an SAPEM job to monitor for the triggering of an SAP event. You can also define box jobs, which are containers that hold other jobs or box jobs. You can define your own job type.

Each job type has required and optional attributes that define the job. The job types have many common attributes and CA Workload Automation AE treats them all similarly. The primary differences between them are the actions taken when the jobs run.

The structure of a job depends on the job type. For example, the following illustration shows the structure of a Command, File Watcher, and Box job:



Common Job Attributes

Some JIL attributes are common to all job types. For example, you can define any job to send an alarm if the job fails or terminates. You can also define starting or restart conditions for any job.

Required Attributes for All Job Types

The following attribute is required for all job types:

- machine

Note: By default, the job type is set to CMD. You can specify a different job type using the job_type attribute.

Optional Attributes for All Job Types

The following attributes are optional for all job types:

- alarm_if_fail
- application
- auto_delete
- auto_hold
- avg_runtime
- box_name
- box_terminator
- condition
- date_conditions
- days_of_week
- description
- exclude_calendar
- group
- job_load
- job_type
- max_run_alarm
- min_run_alarm
- must_complete_times
- must_start_times
- n_retrys
- notification_id
- notification_msg
- owner (This attribute does not apply to File Trigger jobs.)
- permission
- priority
- resources
- run_calendar
- run_window
- send_notification
- service_desk
- start_mins
- start_times
- svcdesk_attr
- svcdesk_desc
- svcdesk_imp
- svcdesk_pri
- svcdesk_sev
- term_run_time
- timezone

Job States

CA Workload Automation AE tracks the current state, or status, of every job. The value of a job's status checks when to start jobs with dependencies on the tracked job. The job status is displayed in the job report generated by the autorep command and in the CA WCC GUI.

A job can have one of the following statuses:

ACTIVATED

Indicates that the top-level box in which the job resides is currently in a RUNNING state but the job has not yet started.

FAILURE

Indicates one of the following:

- For a command job, the command exited with a code greater than the max_exit_success (maximum exit code for success) attribute value specified for the job.
- For a box job, at least one job in the box ended with a FAILURE status or the box_failure (exit condition for box failure) attribute evaluated to TRUE.

By default, any exit code greater than 0 is interpreted as FAILURE. CA Workload Automation AE issues an alarm when a job fails.

INACTIVE

Indicates that the job has not yet been processed. Either it has never run, or its status was intentionally changed to deactivate its previous completion status.

ON_HOLD

Indicates that the job is on hold and will not run until it receives the JOB_OFF_HOLD event.

ON_ICE

Indicates that the job is removed from all conditions and logic, but is still defined. Operationally, this condition is equivalent to deactivating the job, and it remains on ice until it receives the JOB_OFF_ICE event.

PEND_MACH

Indicates that a job can logically start (that is, all the starting conditions have been met), but the machine to which it is assigned is currently offline, either because of a MACH_OFFLINE event or because the computer was proactively put in an inactive state. When the machine comes back online and the required load units become available, CA Workload Automation AE starts the job.

QUE_WAIT

Indicates that the job can logically start (that is, all the starting conditions have been met), but the machines to which it is assigned do not have enough available load units. When the required load units become available, CA Workload Automation AE starts the job.

RESTART

Indicates that the job was unable to start because of hardware or application problems, and has been scheduled to restart.

RESWAIT

Indicates that the job is waiting for a resource before it can continue running. When the resource is available, CA Workload Automation AE starts the job.

RUNNING

Indicates that the job is running. If the job is a box job, the jobs in the box can start (other conditions permitting). If the job is a command or file watcher job, the RUNNING status indicates that the specified process is actually running on the client computer.

STARTING

Indicates that the scheduler has initiated the start job procedure with the agent. This status does not apply to box jobs.

SUCCESS

Indicates that the job exited with a code equal to or less than the max_exit_success (maximum exit code for success) attribute value specified for the job.

If the job is a box, this value means that all the jobs in the box have finished with a SUCCESS status or the box_success (exit condition for box success) attribute evaluated to TRUE.

By default, only exit code 0 is interpreted as SUCCESS. However, you can reserve a range of values up to the max_exit_success value to interpret as SUCCESS for each job.

TERMINATED

Indicates that the job ended while in the RUNNING state. A job can be terminated if it receives a KILLJOB event or if it was defined to terminate if its containing box failed. A job might also be terminated if it exceeds its term_run_time (maximum run time) attribute value or if it receives a UNIX kill command. If the job itself fails, it has a FAILURE status, not a TERMINATED status. CA Workload Automation AE issues an alarm when a job is terminated.

WAIT_REPLY

Indicates that the job is waiting for a user reply before it can continue running.

Notes:

- Jobs that depend on a job that is ON_ICE run as though the job succeeded. Dependent jobs do not run when a job is ON_HOLD.
- If a job's starting conditions are already satisfied when it is taken off hold, it is scheduled to run. However, when a job is taken off ice, it does not run (even if its starting conditions are already satisfied) until its starting conditions recur.

Example: Status of Simple Command Jobs

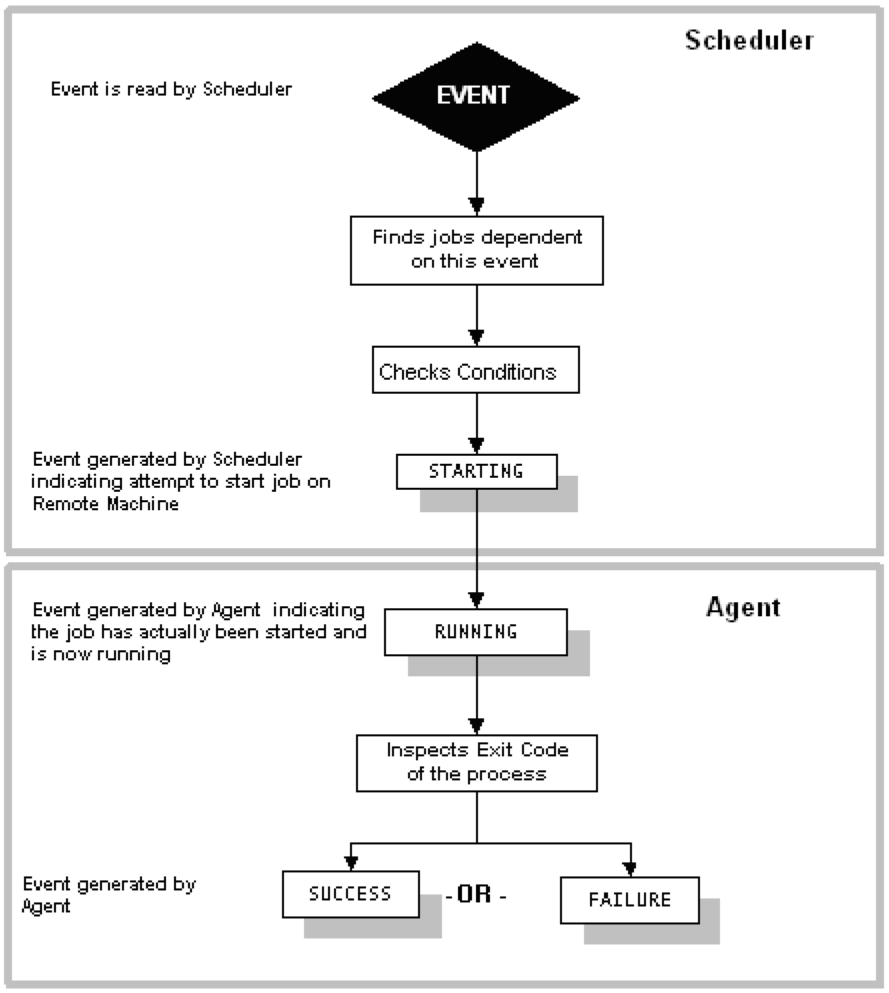
A change in job status is reported as a CHANGE_STATUS event, which the scheduler records in its log when the status is processed. For example, when the job *test_job* changes from STARTING to RUNNING, the scheduler log contains the following entry:

```
EVENT: CHANGE_STATUS STATUS: RUNNING JOB: test_job
```

The following illustration shows the simplest state transition for a job, in which an event satisfies the starting conditions for the job.

The job starts, processes, and completes with either a FAILURE or SUCCESS exit code.

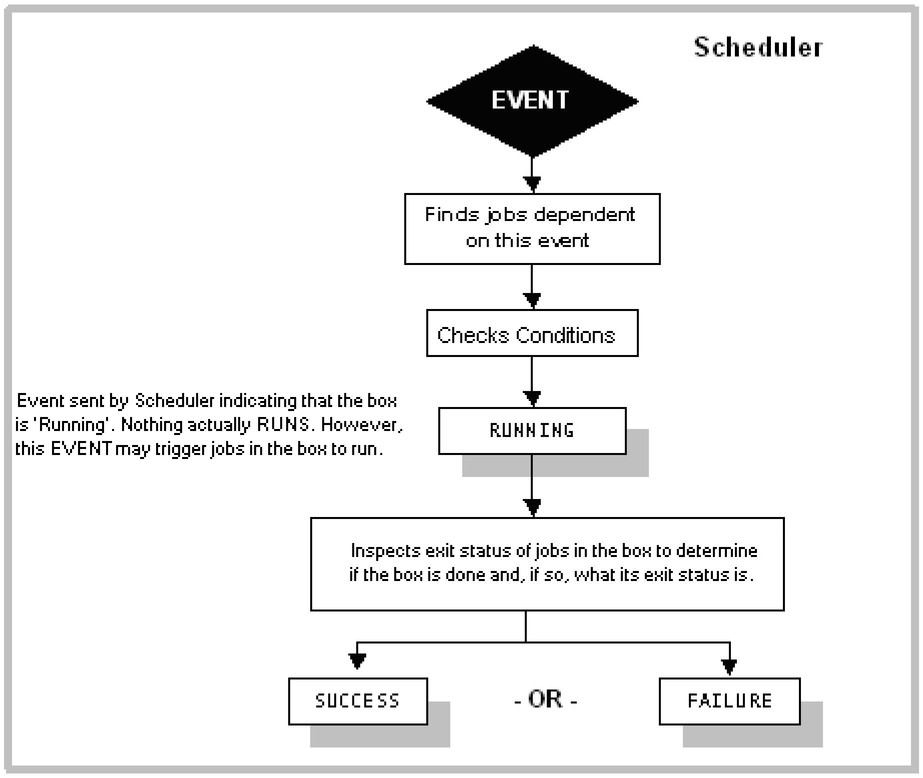
Note: In the following illustration, statuses are shown as shaded boxes:


Scheduler
Event is read by Scheduler
EVENT
Finds jobs dependent on this event
Checks Conditions
Event generated by Scheduler indicating attempt to start job on Remote Machine
STARTING
Agent
Event generated by Agent indicating the job has actually been started and is now running
RUNNING
Inspects Exit Code of the process
Event generated by Agent
SUCCESS
- OR -
FAILURE

Example: Status of Box Jobs

For a job in a box, the job enters the ACTIVATED state when the top-level box in which it resides enters the RUNNING state, as the following illustration shows. After the job starts, the remainder of the scenario is the same as for simple jobs.

Note: In the following illustration, statuses are shown as shaded boxes:


Scheduler
EVENT
Finds jobs dependent on this event
Checks Conditions
Event sent by Scheduler indicating that the box is 'Running'. Nothing actually RUNS. However, this EVENT may trigger jobs in the box to run.
RUNNING
Inspects exit status of jobs in the box to determine if the box is done and, if so, what its exit status is.
SUCCESS
- OR -
FAILURE

A box always enters the RUNNING state as soon as all its starting conditions are met. This RUNNING event usually starts jobs in the box.

If a job has an associated priority, all its starting conditions have been met, and there are not enough machine resources available, it enters the QUE_WAIT state. When resources become available, it enters the STARTING state, and then runs.

Because the job state reflects the scheduler status, a job might have actually completed even if the scheduler has not processed that event. In this case, CA Workload Automation AE still shows the job's status as RUNNING. To view all the events for a job, including those that have not yet processed, display the job detail in the output of the autorep command.

In addition, the job state always reflects the most recent event processed. Therefore, after a job completes, the status remains as it was on completion. If the job ended successfully, the status remains SUCCESS until the job runs again.

Note: When a box job starts, all jobs in the box change their state to ACTIVATED before they run. Jobs run immediately unless other conditions apply. If a box completes before a job runs, the job is set to INACTIVE at box completion. As a result, jobs do not retain their statuses from previous box processing cycles when a new box cycle begins.

More information:

Box Jobs (see page 175)

Insert a Job

Before you can run a job, you must create a job definition and store it in the database. The job definition includes the job name, attributes that describe its intended behavior, and the values of those attributes.

To insert a job

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.
2. Specify the following definition:

```
insert_job: job_name
machine: machine_name
job_type: type
attribute: value
[attribute: value...]
```

job_name

Defines a unique name for the job.

machine_name

Specifies the name of an existing machine definition where the agent runs.

type

Specifies the type of job you are defining.

attribute

Specifies the name of a JIL attribute that applies to the job type that you are defining. You can specify one or more attributes.

Note: For more information about how to define specific job types, see the chapter for that job type. For more information about attributes and their JIL syntax, see the *Reference Guide*.

value

Defines the value of the corresponding attribute.

3. Do *one* of the following:
   - Enter **exit** if you are using interactive mode.
   - Redirect the script to the jil command if you are using a script.

   The data is loaded into the database and the job is defined.

Note: You can also define a job using CA WCC. For more information about using CA WCC, see the CA WCC *Online Help*.

Example: Define a Command Job

This example runs the /bin/touch command on the file named /tmp/test_run.out. The job runs on the UNIX client computer named unixagent.

```
insert_job: test_run
job_type: CMD /* This attribute is optional for Command jobs. CMD is the default. */
machine: unixagent
command: /bin/touch /tmp/test_run.out
```

More information:

Issue JIL in Interactive Mode on UNIX (see page 32)
Issue JIL in Interactive Mode on Windows (see page 33)
Issue JIL Using a Script on UNIX (see page 34)
Issue JIL Using a Script on Windows (see page 35)

Update a Job

You can update an existing job definition.

To update a job

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.

2. Specify the following definition:

   ```
   update_job: job_name
   attribute: value
   [attribute: value...]
   ```

   job_name

   Specifies the name of the job you want to update.

   attribute

   Specifies the name of a JIL attribute that applies to the job type that you are updating. You can specify one or more attributes.

   Note: For more information about specific job types, see the chapter for that job type. For more information about attributes and their JIL syntax, see the *Reference Guide*.

   value

   Defines the value of the corresponding attribute.

3. Do *one* of the following:

   ■ Enter **exit** if you are using interactive mode.

   ■ Redirect the script to the jil command if you are using a script.

   The data is loaded into the database and the job is updated.

Note: You can also update a job using CA WCC. For more information about using CA WCC, see the CA WCC *Online Help*.

More information:

Issue JIL in Interactive Mode on Windows (see page 33)
Issue JIL in Interactive Mode on UNIX (see page 32)
Issue JIL Using a Script on UNIX (see page 34)
Issue JIL Using a Script on Windows (see page 35)

Delete a Job

When you no longer need a job definition, you can delete it from the database.

To delete a job

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.
2. Specify the following subcommand:
   ```
   delete_job: job_name
   ```
   job_name

   Specifies the name of the job you want to delete.
3. Do *one* of the following:
   - Enter **exit** if you are using interactive mode.
   - Redirect the script to the jil command if you are using a script.

   The delete request is issued. When JIL is in job verification mode (the default), the delete_job subcommand scans the ujo_job_cond table and notifies you of any dependent conditions for the deleted job before deleting it.

Note: You can also delete a job using CA WCC. For more information about using CA WCC, see the CA WCC *Online Help*.

Example: Delete a Job

This example deletes the test_run job.

```
delete_job: test_run
```

Running a Job After Using JIL

After you submit a job definition to the database, it runs according to the starting parameters specified in its JIL script. That is, the scheduler continually polls the database, and when it verifies that the starting parameters are met it runs the job.

If a JIL script does not specify any starting parameters for a job, the scheduler does not start the job automatically; the job starts only if you issue the sendevent command.

Note: For more information, see the *Reference Guide*.

Example: Run a Job with the sendevent Command

This example assumes that a job named test_install has no starting parameters specified in its JIL script. The only way to start it is to issue the following command:

```
sendevent -E STARTJOB -J test_install
```

This command tells the scheduler to start the job named test_install.

Specify the Job Owner

By default, the owner attribute is set to *authenticated_user@host*, where *authenticated_user* is the operating system user who invokes jil to define the job. If your job requires a different user ID, you must override the owner value. Otherwise, the job does not run successfully.

For jobs that run on other software, such as PeopleSoft and databases, the owner can be the user ID associated with and authenticated by the software. For example, SAP jobs run under SAP user IDs. As another example, an FTP job requires an FTP user ID to connect to the FTP server.

You must define user IDs and their corresponding passwords to CA Workload Automation AE by using the autosys_secure command. In a job definition, you can use the owner attribute to specify the user ID. CA Workload Automation AE retrieves the corresponding password from the database.

To specify the job owner, add the owner attribute to your job definition.

Notes:

- The owner attribute does not apply to File Trigger jobs.
- If CA Workload Automation AE is running in native security mode, you can change the owner value only if you have EDIT superuser permissions.
- If CA Workload Automation AE is running in external security mode using CA EEM, you can change the owner value only if you have as-owner authority.
- CA Workload Automation AE uses the owner value for all job types except for File Trigger. CA Workload Automation AE does *not* use the oscomponent.default.user parameter located in the agent's agentparm.txt file.
- For more information about the owner attribute, see the *Reference Guide*.

Example: Specify a Job Owner

Suppose that CA Workload Automation AE is running in external security mode and you have as-owner authority as defined in CA EEM. You can specify the owner attribute in job definitions. The following job runs under the prod user on the unixagent computer:

```
insert_job: jobA
job_type: CMD
machine: unixagent
command: /bin/touch /tmp/test_run.out
owner: prod@unixagent
```

Custom Calendars

You can use the autocal_asc utility to define any number of custom calendars, each with a unique name and containing any number of dates or date/time combinations. You can use these calendars to specify days on which to run the jobs with which they are associated or to specify days on which jobs with which they are associated should not run. Calendars exist independently of any jobs associated with them and are referenced by jobs through job definitions.

Example: Define a Standard Calendar

The following is a calendar definition for the standard calendar, Holiday2006. This calendar can be used with jil attributes run_calendar or exclude_calendar as well as in an extended calendar definition.

```
calendar: Holiday2006
01/01/2006 00:00
01/02/2006 00:00
01/16/2006 00:00
02/20/2006 00:00
05/29/2006 00:00
07/02/2006 00:00
09/04/2006 00:00
11/23/2006 00:00
11/24/2006 00:00
12/25/2006 00:00
12/26/2006 00:00
```

This example schedules jobs to run everyday of the week at 2:00 p.m., except for the days listed in the Holiday2006 calendar.

```
insert_job:JobWithExcludeCalendar
command:ls
machine:localhost
date_conditions:1
days_of_week:all
start_times:”14:00”
exclude_calendar:Holiday2006
```

Global Variables

You can define global variables using the sendevent command. After you define a global variable to CA Workload Automation AE you can use the variable as a job dependency. The job dependency is satisfied only when the value of the expression evaluates to TRUE.

Note: For more information about using the sendevent command to define global variables, see the *Reference Guide*.

Example: Define a Global Variable

This example sets the global variable "today" to a value of “12/25/2007":

sendevent -E SET_GLOBAL -G "today=12/25/2007"

More Information:

Job Dependencies Based on Global Variables (see page 102)

Alerts

You can define the following job types to monitor a condition continuously:

- CPU Monitoring (OMCPU)
- Database Monitor (DBMON)
- Database Trigger (DBTRIG)
- Disk Monitoring (OMD)
- File Trigger (FT)
- Text File Reading and Monitoring (OMTF)
- Windows Event Log Monitoring (OMEL)
- Windows Services monitoring (OMS)

Each time the specified condition occurs, an ALERT event is written to the scheduler log file (event_demon.$AUTOSERV on UNIX and event_demon.%AUTOSERV% on Windows). An alert helps you track and report each time that a monitored condition occurs.

Note: An alert is only generated for triggers that occur during continuous monitoring. Alerts are not generated for non-continuous monitoring (NOW and WAIT).

To stop a continuous monitor, you must complete the job manually by issuing the following command:

```
sendevent –E KILLJOB –J job_name
```

For non-continuous monitors the proper event order will be reflected as

1. STARTING
2. RUNNING
3. SUCCESS

For a continuous monitor the event order will be reflected as

1. STARTING
2. RUNNING
3. ALERT
4. ALERT
5. ALERT...
6. KILLJOB
7. TERMINATED
8. JOBFAILURE (ALARM)

You can view the text of alerts using the following methods:

- View the scheduler log file
- Issue the following command:

  `autorep –J` *`job_name`* `–d`
- CA WCC

Note: You cannot manually send the ALERT event using the sendevent command.

Example: Trigger Alerts When Monitoring CPU Usage Continuously

This example continuously monitors used CPU on the unixagent computer. When the job runs, it goes into a RUNNING status. When the job detects that the used CPU is within 70 and 100 percent, an ALERT event is raised (an alert is written to the scheduler log file). The available, used CPU and load averages will be reported as part of the ALERT event and the status message is reported with the RUNNING event. Subsequently, each time the job detects that the use CPU meets the monitored condition, an alert is triggered. The job only ends when it is complete manually.

```
insert_job: cpu_monitoring_used
job_type: OMCPU
machine: unixagent
lower_boundary: 70
cpu_usage: USED
inside_range: TRUE
monitor_mode: CONTINUOUS
```

In contrast, the following example monitors used CPU in WAIT monitor mode. When the job runs, it goes into a RUNNING status. When the job detects that the used CPU is within 70 and 100 percent, the job completes. No alert is triggered. The available, used CPU and load averages are reported on the SUCCESS event.

```
insert_job: cpu_monitoring_used_wait
job_type: OMCPU
machine: unixagent
lower_boundary: 70
cpu_usage: USED
inside_range: TRUE
monitor_mode: WAIT
```

Starting Conditions

CA Workload Automation AE verifies if it should start a job based on the evaluation of the starting conditions defined for the job. All defined starting conditions must be *true* for a job to start. These conditions can include one or more of the following:

- Date and time scheduling parameters are met.
- Starting conditions specified in the job definition evaluate to TRUE.
- For jobs in a box, the box is in the RUNNING state.
- The current job state is not ON_HOLD or ON_ICE.
- The job's machine is not OFFLINE.

When an event changes any of the above conditions, CA Workload Automation AE finds all the jobs that might be affected by this change and checks if it can start them.

Date and Time Dependencies

You can use JIL statements to schedule CA Workload Automation AE jobs to start at a specific date and time. CA Workload Automation AE then calculates a matrix of specified day, date, and time values and starts jobs accordingly. A time range cannot span more than 24 hours.

For example, you can define a job to start on Monday, Wednesday, and Friday at 8:00 a.m. and 5:00 p.m.

You can specify days of the week or actual dates, but you cannot specify both. You can specify days of the week using JIL, but you can only specify actual dates using custom calendars. You can also specify a time zone to apply to your starting times, and you can define a job to start at one specific time of day or hourly, denoted in minutes past the hour.

TZ Environment Variable

Valid on UNIX

By default, jobs with time-based starting conditions that do not specify a time zone are scheduled to start based on the time zone of the TZ environment variable (that is, the time zone under which the scheduler runs).

Before you start the scheduler, ensure that the TZ environment variable is set. The scheduler must be started once after you upgrade your database to insert the value of the TZ environment variable into the database. Do this before executing jil or autorep.

Job Dependencies Based on Job Status

You can define starting conditions to start jobs based on the current status of one or more jobs that exist in the database. In this way you can program simple or complex prerequisites for starting a job.

For example, you can implement a single-threaded, batch queue-like set of job dependencies so that JobB starts when JobA achieves a SUCCESS status and JobC starts when JobB achieves a SUCCESS status.

You can configure more complex conditions by combining a series of conditions with the AND and OR logical operators. You can use the pipe symbol (|) instead of the word OR and the ampersand symbol (&) instead of the word AND. Spaces between conditions and delimiters are optional. You can specify even more complex conditions by grouping the expressions in parentheses, which force precedence. The equation is evaluated from left to right.

For example, in the following set of starting conditions, either both A and B must be successful or both D and E must be successful for the statement to evaluate as TRUE:

```
(success(JobA) and success(JobB)) or (success(JobD) AND success(Job E))
```

Note: If you specify a condition for an undefined job, the condition evaluates as FALSE, and any jobs dependent on this condition do not run. You can use the job_depends command to check for this type of invalid condition statement.

The syntax for defining job dependencies is the same whether the job is being defined using JIL or the CA WCC GUI, except that the JIL statement begins with the JIL condition keyword.

The following is the syntax for conditions based on job status:

`status(job_name)`

status

Indicates the status as one of the following:

success

Indicates that the status condition for *job_name* is SUCCESS. You can abbreviate this value to s.

failure

Indicates that the status condition for *job_name* is FAILURE. You can abbreviate this value to f.

done

Indicates that the status condition for *job_name* is SUCCESS, FAILURE or TERMINATED. You can abbreviate this value to d.

terminated

Indicates that the status condition for *job_name* is TERMINATED. You can abbreviate this value to t.

notrunning

Indicates that the status condition for *job_name* is anything except RUNNING. You can abbreviate this value to n.

job_name

Identifies the job on which the new job is dependent.

You can also abbreviate the dependency specification EXIT CODE to e and VALUE (of a global variable) to v.

You can use the max_exit_success (maximum exit code for success) attribute set for a job to control the value of the SUCCESS status. If you specify this attribute, any job that exits with an exit code less than or equal to the specified value is treated as a success. A FAILURE status means the job exited with an exit code higher than this value. The default exit code for normal job completion is 0. A TERMINATED status means the job was killed.

Note: You can use either uppercase or lowercase letters to specify a status. However, you cannot use mixed case.

Example: Job Dependencies

For a job that runs only when the job named DB_BACKUP succeeds, you would specify the job dependency as follows:

```
success(DB_BACKUP)
```

If JobC should only start when both JobA and JobB complete successfully or when both JobD and JobE complete (regardless of whether JobD and JobE failed, succeeded, or terminated), you would specify the following dependency in the job definition for JobC:

```
(success(JobA) AND success(JobB)) OR (done(JobD) AND done(JobE))
```

As indicated in this example, you can use any job status as part of the specification for a specific job's starting conditions. With this latitude, you can program branching paths that must be taken and provide alternate actions for error conditions.

For example, if JobB fails after partially processing, you might want to call a routine called Backout that reverses the changes that were made. You would specify the following job dependency in the job definition for Backout:

```
failure(JobB)
```

You can use the notrunning operator to keep multiple jobs from running simultaneously. For example, assume you do not want to run a database dump (DB_DUMP) and a file backup (BACKUP) at the same time because such processing would adversely impact performance. However, you might have a smaller job that can run as long as both of these resource-intensive jobs are not running. You would specify the smaller job's dependency as follows:

```
notrunning(DB_DUMP) AND notrunning(BACKUP)
```

Managing Job Status

Starting conditions that are based on job status use the current (or most recent) completion status of the job. The current completion status is defined by the job run, regardless of when that run occurred.

However, if you want to enforce the concept of time-based processing cycles, where the completion status of a job for some previous time period should not affect the processing of this time cycle, there are several options available.

When a box job starts, the status of all the jobs in the box changes to ACTIVATED. Therefore, subsequent jobs in the box that depend on the completion of jobs performed earlier in the same box only use the completion statuses from this box run. Placing the jobs in one processing cycle inside a top-level box and setting the box to start at the beginning of the processing cycle prevents time-critical jobs from being affected by invalid information.

When a job is first entered into the database, and before it runs for the first time, its status is set to INACTIVE. By changing the status of jobs that have completed but whose completion status should no longer be used in dependent job conditions to INACTIVE, the completion status from the last run is no longer the current status and it is not used.

Use the sendevent command to change a job status to INACTIVE. Alternatively, you could create a CA Workload Automation AE job to accomplish this. If you change the status of a top-level box to INACTIVE, all the jobs in the box also change to INACTIVE.

Deleting and reinserting the job using JIL accomplishes the same thing. However, the past reporting history on the job is no longer available. Updating a job using JIL does not change the status of the job.

Job Dependencies Based on Exit Codes

You can use the following syntax to base job dependencies on exit codes that indicate completed tasks. In this way, you can implement even more specific branching logic for recovering from job failures.

This method of defining job dependencies has the following format:

```
exitcode (job_name) operator value
```

job_name

Defines the name of the job upon which the new job depends.

operator

Specifies one of the following exit code comparison operators:

=

Equal to.

!=

Not equal to.

<

Less than.

>

Greater than.

<=

Less than or equal to.

>=

Greater than or equal to.

value

Defines the numeric exit code value on which to base the dependency.

For example, if a broken communication line results in JobA failing with an exit code of 4, and you want the system to run a script (JobB) that redials the line when this code is encountered, you would enter the following for the job dependency specification for the JobB redial job:

```
exitcode (JobA) = 4
```

You can use any job status or exit codes as part of the specification for starting conditions. You can abbreviate the dependency specification exitcode with the letter e (uppercase or lowercase).

Exit Codes and Batch Files in Jobs Running on Windows

When you define jobs to run batch files on Windows, you should be aware of and account for Windows-specific behavior.

Windows programs return any exit values that are programmed in the executable code. This exit value is the last thing returned to Windows when the program terminates.

Generally, a zero (0) exit code indicates success, while a non-zero exit code indicates an error. The expected error values should be documented with each individual program, but some programs can return unexpected exit codes. Modify these programs so that they return expected values, and use these values when specifying exit code dependencies.

Jobs are created using standard Windows process creation techniques. After the job is created, the agent waits for the job to complete. When the job completes, CA Workload Automation AE gets the program exit code from Windows and stores it in the database for later use.

When launching programs directly, the exit codes are returned and put in the database. However, there are some exit code behaviors that you must take into consideration when using a job to start *.BAT batch files.

The exit code returned from a batch file is the return code from the last operation executed in that particular batch file. Consider the following example:

```
REM test batch file
test
if errorlevel 1 goto bad
goto good
:bad
del test.tmp
:good
exit
```

This sample batch file returns a 0 exit code when the test program exits with a 1 exit code as long as test.tmp exists. If test.tmp does not exist, the return code is from the del line and not from the line that runs the test. Therefore, this batch file returns a 0 (successful) exit code, even if test failed to execute as intended.

To help handle situations like this, CA Workload Automation AE supplies a program called FALSE.EXE. This program resides in the Windows %AUTOSYS/bin directory and takes only one parameter, which is the exit code you want FALSE.EXE to return on completion. You can use FALSE.EXE as follows:

```
REM test batch file
test
if errorlevel 1 goto bad
exit
:bad
del test.tmp
false 1
```

When test fails with error level 1, this batch file returns an exit code of 1 from FALSE.EXE, whether the test.tmp file exists or not.

Job Dependencies Based on Global Variables

You can base job dependencies on global variables set using the sendevent command. When using global variables in this way, the job dependency is satisfied only when the value of the expression evaluates to TRUE.

This method of defining job dependencies has the following format:

`VALUE(global_name) operator value`

global_name

Defines the name of the global variable upon which the job depends.

Limits: This value can be up to 30 characters in length. The following characters are valid: a-z, A-Z, 0-9, period (.), underscore (_), and hyphen (-). You can include spaces in a global variable name.

operator

Specifies one of the following exit code comparison operators:

=

Equal to.

!=

Not equal to.

<

Less than.

>

Greater than.

<=

Less than or equal to.

>=

Greater than or equal to.

value

Defines the numeric or text value of the global variable on which to base the dependency.

Limits: This value can be up to 30 characters in length and cannot contain quotation marks or spaces. The following characters are valid: a-z, A-Z, 0-9, period (.), underscore (_), and hyphen (-).

Note: When using JIL, use the condition attribute to enter the above expression in the appropriate JIL script.

For example, assume that a set of jobs in a box should only run with a manager's approval. In this case, use the following syntax to set the global variable named manager-ok to OK, and make the top-level box job dependent on this global variable:

```
VALUE(manager-ok) = OK
```

You can abbreviate the dependency specification VALUE with the letter v (uppercase or lowercase).

Starting Conditions and Boxes

When you put a job in a box, it inherits all of the starting conditions of the box. Therefore, all starting conditions defined for the box must be met and the box must enter the RUNNING state before the job can run. If there are no additional conditions on the job, it starts as soon as the box starts. A job runs only once for each box execution.

By default, there is no sequential job processing in a box. For example, if three jobs are in a box, all three jobs start when the box starts if they have no additional conditions.

To implement a processing sequence for jobs in a box, you must specify additional starting conditions for each job. For example, you could specify that Job1 has no starting conditions, Job2 depends on the completion of Job1, and Job3 depends on the completion of Job2.

Note: Jobs that depend on a job that is ON_ICE run as if that starting condition has been satisfied. In this scenario, if Job2 enters the ON_ICE state, then Job1 and Job3 start simultaneously when the box they are in starts running.

More Information:

Global Variables (see page 93)

Job Run Numbers and Names

CA Workload Automation AE uses run numbers for jobs. The *run number* is a unique integer associated with every run of a job.

Consecutive run numbers are assigned every time a top-level job starts. A top-level job is a job that is not contained in a box, and these run numbers are inherited by every job in a box. This means that all jobs in a top-level box have the same run number as the number used for the run of the box. This design permits runs of nested jobs to be associated together in the same run.

If a job restarts, the run number remains the same and the ntrys field is incremented. In the standard reports (autorep command), these two values are displayed in the run column as run_num/ntry.

The run_num/ntry value is defined in the run-time environment for the job, and is accessible to shell scripts or executables run as the job's command. This value is contained in the variable $AUTORUN.

CA Workload Automation AE also maintains a value for each job's name, which is defined in the runtime environment for the job.

As with $AUTORUN, this value is accessible to shell scripts or executables run as the job's UNIX command. The value is contained in the variable $AUTO_JOB_NAME.

On Windows, the environment variables are %AUTORUN% and %AUTO_JOB_NAME%.

How Time Dependencies Are Set

If you do not define starting conditions for a job, it only runs when you issue a sendevent command for it. You can set time dependencies for a job so that it runs automatically on specific days and at specific times.

Note: For more information, see the *Reference Guide*.

Example: Set Time Dependencies for a Job

This example shows how to use the date_conditions, days_of_week, and start_times attributes to set time dependencies for a job.

To set the existing job test_run to run automatically on certain days at a certain time (such as 10:00 a.m. and 2:00 p.m. on Mondays, Wednesdays, and Fridays), you could modify the job using the following JIL script:

```
update_job: test_run
date_conditions: y
days_of_week: mo, we, fr
start_times: "10:00, 14:00"
```

This JIL script instructs CA Workload Automation AE to do the following:

- Update the job named test_run.
- Activate the conditions based on date.
- Set the job to run on Mondays, Wednesdays, and Fridays.
- Start the job at 10:00 a.m. and 2:00 p.m. on each of the specified days.

The times shown in the sample script are surrounded by quotation marks because they contain a colon. You can also use a backslash (\) as an escape character for the colon, as the following example shows:

```
start_times: 10\:00, 14\:00
```

Note: If a job runs daily at the same time (for example, 12:00) and you edit the job definition and save it at 11:59, the job will not run until the next day at 12:00.

When you save a start time job definition to the database within one minute of the specified start time, the start time is placed in the future (that is, tomorrow). However, if the start time is two minutes or more from the save time, the job runs at the next occurrence of the specified start time (that is, today).

Example: Base Time Settings on a Specific Time Zone

Use the timezone attribute to base the time settings for a job on a specific time zone. If you specify a time zone that includes a colon, you must surround the time zone name with quotation marks, as in the following example:

```
timezone: "IST-5:30"
```

Example: Run a Job Every Day

To run the job every day, instead of only on specific days, specify the all value instead of listing the individual day values. For example:

```
days_of_week: all
```

Example: Schedule a Job to Run on Specific Dates

To schedule the job for specific dates, instead of specific days of the week, specify a custom calendar. Use the autocal_asc command to define the calendar, and then use the run_calendar attribute to specify the calendar name (for example, weekday_cal) in the job definition. For example:

```
run_calendar: weekday_cal
```

Example: Exclude a Job from Running on Specific Dates

To specify a custom calendar that defines the days on which the job should not run, use the autocal_asc command to define the calendar, and use the exclude_calendar attribute to specify the calendar name (for example, holiday_cal) in the job definition. For example:

```
exclude_calendar: holiday_cal
```

Example: Schedule a Job to Run at Specific Times Every Hour

To run the job at specific times every hour instead of at specific times of the day, use the start_mins attribute to specify the minutes past every hour that the job should run. For example, to run a job at 15 minutes after and 15 minutes before each hour, add the following statement to the job definition:

```
start_mins: 15, 45
```

Dependent Jobs

Jobs can be dependent on the successful completion of other jobs. The only difference between a dependent job and a simple job is its dependency on another job. To define job dependencies, specify the condition attribute in the job definition.

CA Workload Automation AE lets you specify a time limit in the condition attribute that applies in job dependency evaluations. The job's execution environment is verified exclusively by the profile, which is sourced immediately before the job starts. On UNIX, by default the /etc/auto.profile file on the client computer is sourced. On Windows, the variables set by the installer in the agent’s profile directory are sourced or set. You can use the profile attribute to override the default profile.

Note: For more information about the condition attribute, see the *Reference Guide*.

Example: Create a Dependent Command Job

This example shows a JIL script that defines a dependent command job named EOD_post. EOD_post depends on the successful completion of the File Watcher job named EOD_watch.

```
insert_job: EOD_post
job_type: cmd
machine: prod
condition: success(EOD_watch)
command: $HOME/POST
```

This JIL script instructs CA Workload Automation AE to do the following:

- Add a new job named EOD_post.
- Define the job as a command job.
- Run the job on the client computer named prod.
- Run the job only if the file watcher job named EOD_watch completes with a SUCCESS status.
- Source the /etc/auto.profile file (CA Workload Automation AE sources this file by default), and run the job named POST located in the job owner's home directory.

More information:

Job Profiles (see page 115)

Look-Back Conditions

CA Workload Automation AE supports look-back conditions. You can use look-back conditions to base dependencies for a job on the last run of another job. The *last run* is defined by the ending time of the last successful run of a job. If the job has run with the specified result, the condition or predecessor is satisfied and the job starts. If not, the condition is not satisfied and the job for which the look-back condition is defined does not start.

To specify a look-back dependency, enter the job name followed by a comma (,) then *HH* (hours), period (.) and *MM* (minutes).

Example: Specifying Look-Back Conditions

This example shows a job definition with look-back conditions.

In the following job definition, the command job test_sample_04 can only start if all of the following conditions are met:

- The last run of test_sample_01 completed successfully during the last 12 hours.
- The last run of test_sample_02 completed with a FAILURE status during the last 24 hours.
- The last run of test_sample_03 completed successfully at any time.

```
insert_job: test_sample_04
machine: localhost
command: sleep 10
condition: success(test_sample_01,12.00) AND failure(test_sample_02,24.00) AND
success(test_sample_03)
```

Specifying One-Time Job Overrides

You can use the override_job subcommand to specify an override that changes the behavior of a specific job during its next run. Job overrides are applied only once. If a RESTART event is generated because of system problems, CA Workload Automation AE reissues a job override until the job actually runs once, or until the maximum number of retries limit is met. After this, CA Workload Automation AE discards the override.

You can modify the following attributes in a job override:

- auto_hold
- command
- condition
- date_conditions
- days_of_week
- exclude_calendar
- machine
- max_run_alarm
- min_run_alarm
- n_retrys
- profile
- run_calendar
- run_window
- start_mins
- start_times
- std_err_file
- std_in_file
- std_out_file
- term_run_time
- watch_file
- watch_file_min_size
- watch_interval

JIL will not accept an override if it results in an invalid job definition. For example, if a job definition has only one starting condition, start_times, JIL will not let you set the start_times attribute to NULL because removing the start condition makes the job definition invalid (no start time could be calculated).

One-time job overrides are applied to jobs restarted due to system problems, but are not applied to jobs restarted because of application failures.

System problems include the following:

- Machine unavailability
- Media failures
- Insufficient disk space

Application failures include the following:

- Inability to read or write a file
- Command not found
- Exit status greater than the defined maximum exit status for success
- Various syntax errors

How Job Overrides Are Set

To set job overrides, use the override_job subcommand to specify the job and attributes to override. You can also temporarily delete a job attribute in this manner.

Example: Define a One-time Override for a Job

This example shows how to define a one-time job override. The following script runs the job RunData with no conditions (where some had been previously specified) and outputs the results to a different output file:

UNIX:

```
override_job: RunData
condition: NULL
std_out_file: "tmp\SpecialRun.out"
```

Windows:

```
override_job: RunData
condition: NULL
std_out_file: "C:\tmp\SpecialRun.out"
```

Example: Cancel a Job Override Before it Runs

This example shows how to cancel a job override before it runs. To cancel overrides for a job, enter the override_job subcommand followed by the job name and the delete parameter. For example:

```
override_job: RunData delete
```

Note: After you submit a JIL script to the database, you cannot view the script or edit an override. To change the override values, you must submit another JIL script with new values or use the CA WCC Quick Edit. However, the original override remains stored in the ujo_overjob table in the database.

Date and Time Attributes and Time Changes

Your operating system might automatically change the system clock to reflect the switch to either standard time (ST) or daylight time (DT), and the scheduling of time-dependent CA Workload Automation AE jobs might shift to adjust for the time change. Jobs that are not time-dependent run as appropriate.

There are two types of time dependencies: *absolute* and *relative.*

Jobs with absolute time dependencies are defined to run at a specific time of the day (for example, 9:30 on Thursday or 12:00 on December 25). The following attributes define absolute time dependencies:

- days_of_week
- exclude_calendar
- must_start_times
- must_complete_times
- run_calendar
- run_window
- start_times

Relative time dependencies are based either on the current time or relative to the start of the hour (for example, start a job at 10 and 20 minutes after the hour, or terminate a job after it has run for 90 minutes). The following attributes define relative time dependencies:

- auto_delete
- max_run_alarm
- min_run_alarm
- must_start_times
- must_complete_times
- start_mins
- term_run_time
- watch_interval

During the time change, absolute time attributes behave differently than relative time attributes.

Daylight Time Changes

Because the clock loses an hour during the change from standard time to daylight time in the spring, CA Workload Automation AE cannot schedule any jobs using time-dependent attributes during that time.

The solution is to schedule jobs with absolute time dependencies for the missing hour to start during the first minute of the next hour. In this case, because the time change automatically occurs at 2:00 a.m., a job scheduled to run on Sundays at 2:05 runs at 3:00:05 that day; a job scheduled to run every day at 2:45 runs at 3:00:45. Although it might not be possible to start a large number of jobs during the first minute of the hour, this feature does preserve the scheduling order.

If you schedule a job to run more than once during the missing hour (for example, at 2:05 and 2:25), only the first scheduled job run occurs. Additional start times for the same job in the missing hour are ignored.

Jobs with relative time dependencies run as expected. For example, a job specified to run at 0, 20, and 40 minutes after the hour is scheduled for 1:00 ST, 1:20 ST, 1:40 ST, 3:00 DT, 3:20 DT, and 3:40 DT.

Run windows are treated differently. When the specified end of the run window falls during the missing hour, CA Workload Automation AE recalculates its end time, so that the effective duration of the run window remains the same. For example, the product recalculates a run window of 1:00 - 2:30 so that the window ends at 3:30 and the run window still remains open for 90 minutes.

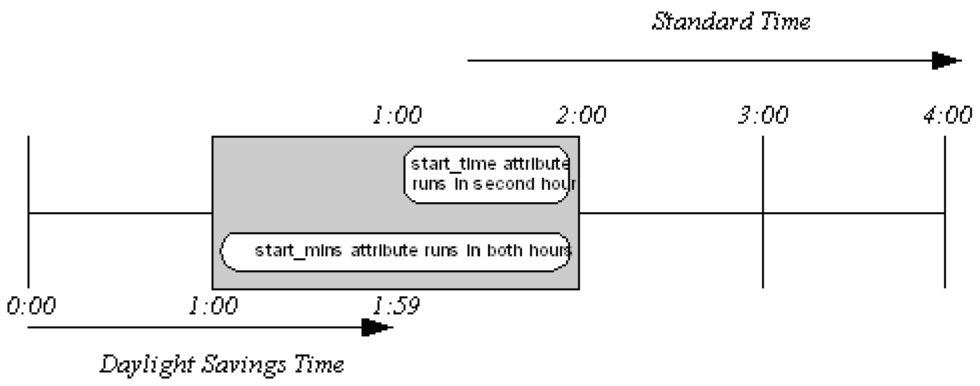
When the run window’s specified start time falls during the missing hour, CA Workload Automation AE moves the start time to 3:00. The end time does not change, so the run window is shortened. For example, a run window of 2:45 - 3:45 becomes 3:00 - 3:45, shortening the run window by 15 minutes.

When the run window’s start and end time both fall during the missing hour, CA Workload Automation AE moves the start time to the first minute after 3:00 and the end time to one hour later. Therefore, the resulting run window might be lengthened. For example, a run window of 2:15 - 2:45 becomes 3:00 - 3:45, or 15 minutes longer.

Standard Time Changes

Because the clock gains one hour during the change from daylight time to standard time in autumn, there are two 1:00-1:59 hours. CA Workload Automation AE only runs jobs for which the start_time attribute is set to between 1:00 and 1:59 during the second (standard time) hour. Jobs for which the start_mins attribute is set run in both hours.

For example, a job scheduled to run on Sundays at 1:05 runs only at the second 1:05. A job scheduled to run every 30 minutes runs at 1:00 DT and 1:30 DT, then again at 1:00 ST and 1:30 ST, and so on, as the following illustration shows:


Standard Time
1:00
2:00
3:00
4:00
start_time attribute runs in second hour
start_mins attribute runs in both hours
0:00
1:00
1:59
Daylight Savings Time

Jobs that are not time-based but have other dependencies still run during the first hour.

Jobs with relative time dependencies run as expected. For example, if a job is scheduled to run on Sunday at 0:30 and its term_run_time attribute is set to 120 minutes, the job would normally terminate at 2:30. On the day of the autumn time change, the job terminates at 1:30 standard time, which is 120 minutes after the job started.

When testing how the change from daylight time to standard time affects your jobs, you must set the system clock to a time before 1:00 a.m. and allow the entire hour to pass before you can observe the time change. If you manually set the time to a period during the 1:00 a.m. to 2:00 a.m. window, the system assumes that the time change has already occurred and does not reset at 2:00 a.m.

Run windows are treated differently. When the specified start of a run window is before the time change and its specified end occurs during the repeated hour, the run window closes during the daylight time period (the first hour). For example, a run window of 11:30 - 1:30 ends at 1:30 DT, not 1:30 ST (that is, the run window remains open for its specified two hours, not for three hours). A problem might occur if there are also associated start times for the job that occur during the repeated hour. If the job in our example also had a start time of 1:15, the start time would be calculated for 1:15 ST and the job would not run on the day of the time change.

When the specified opening of the run window falls during the repeated hour, CA Workload Automation AE moves its start time to the second, standard time hour. The end time does not change, so the length of the run window remains the same. For example, a run window of 1:45 - 2:45 becomes 1:45 ST - 2:45 ST.

When both the specified start and end of the run window occur during the repeated hour, the run window opens during the second, standard time hour.

Job Profiles

A *job profile* defines the non-system environment variables for a job. When you define a job, you can assign a job profile to it. Only one profile can be sourced for a job.

Job profiles apply to the following job types:

- Command jobs
- File Watcher (FW) jobs that are submitted to the legacy agent

On UNIX, job profiles are shell scripts that typically include the definitions and exports of environment variables. The command that the job runs can reference these variables. You can store job profiles on UNIX in any directory. Job profiles on UNIX are always sourced using the job owner's default shell, which is set for the user in the etc/passwd file. Therefore, when you create a job profile, you must use the syntax of the owner's default shell. For example, if the owner's default shell is the Korn shell, you must use Korn syntax in the profile script.

On Windows, you create job profiles using the Job Profiles window in the CA Workload Automation AE Administrator utility. These profiles contain variable=value pairs that define the environment variables. The profiles are stored in the SystemAgent*agent_name*\profiles directory of the CA Workload Automation AE computer. If you move a job profile to another location, you must specify the full path when you assign the profile to a job. The agent uses the variable=value pairs in the job profile to set the environment variables.

Environment Variables

System environment variables are automatically set in the environment of a job. However, user environment variables are not automatically set. You must define all other required environment variables using one or both of the following methods:

- envvars attribute in the job definition
- Job profile

If a job profile is assigned to a job, the agent sources the profile before running the job. When the agent reads the profile, the environment variables in the profile are expanded. For example, if Path is a variable in the profile, the following occurs:

- Any environment variables specified in the value of Path are expanded.
- That path is used to search for the command.
- The new value for the %Path% variable is set before running the command.

If you want to specify the full path name, you can use variables set from the job profile in the path name specification.

The agent reads profile variables in alphabetical order. Therefore, if you plan to expand variables in the profile itself, you must define the variables so that they are in the appropriate order when read alphabetically.

Notes:

- On the agent, you can define environment variables that apply to all jobs at a global agent level, scheduling manager level, or user level. For example, suppose that you want to set an environment variable for all jobs that run on an agent under a specific user (owner). Instead of defining that variable in every job definition using the envvars attribute or in a job profile, you can define the variable on the agent using the oscomponent.environment.variable_user_*userid* parameter. For more information about setting environment variables on the agent, see the *CA Workload Automation Agent for UNIX, Linux, or Windows Implementation Guide*.
- For more information about the envvars and profile JIL attributes, see the *Reference Guide*.

How the Environment for a Job is Sourced

System environment variables are automatically set in the environment for a job. When the job is submitted, the agent processes the following additional information to source the environment, in the following order:

1. Variables set during the CA Workload Automation AE installation.
   - On UNIX the auto.profile file is automatically created during CA Workload Automation AE installation and contains variable definitions such as AUTOUSER. This file is located on the CA Workload Automation AE computer.
   - On Windows variables are set by the installer in the agent's profile directory. The WAAE.txt file contains a set of variables for the product, and an <*instance_name*>.txt file contains a set of variables for each instance. Typically, these files contain the %AUTOROOT%, %AUTOSYS%, %AUTOUSER%, and %PATH% variables.
2. Environment variables defined using the envvars attribute in the job definition (if specified)
3. The job profile defined using the profile attribute (if specified)

Note: The environment variables are set before the job profile variables. Therefore, you can reference system environment variables in job profiles. However, if a variable is set more than once, the last value read is used.

Create a Job Profile

You can create a job profile to define the non-system environment variables that must be set for a Command job or for a File Watcher (FW) job that is submitted to a legacy agent.

To create a job profile, do one of the following:

- On UNIX, create a shell script file that contains the environment variables you want to source.

  You must use the syntax for the default shell of the job owner. You can store this script in any directory.
- On Windows, use the Job Profiles window in the CA Workload Automation AE Administrator utility.

  The job profile is stored in the SystemAgent*agent_name*\profiles directory of the CA Workload Automation AE computer.

Note: For more information about creating, viewing, and deleting job profiles on Windows, see the *Online Help* for the Administrator utility.

Assign a Job Profile to a Job

You can assign a job profile to a job to source the non-system environments that must be set before the job can run.

To assign a job profile to a job

1. Do *one* of the following:
   - Define a Command job (see page 207).
   - Define a File Watcher job on the legacy agent (see page 281).
2. Add the following attribute to the job definition:

   profile: *path_name*

   Specifies a profile that defines the non-system environment variables for the job.

   Notes:

   - On UNIX, specify the name of the profile script. Alternatively, you can specify *path*/*profile_name*.
   - On Windows, specify the name of the profile that you created in the Administrator utility. Alternatively, you can specify *path*\profile_name.
   - You can specify both the computer name and the profile name, which lets you run the job on one computer while using a job profile defined on another computer. For example, you can specify the following path on Windows: \\machine_name\share_name\*profile_name*.txt
3. Run the job.

   The job profile is assigned to Command or File Watcher job.

Notes:

- Job profiles are instance-specific. You cannot assign a profile defined in one CA Workload Automation AE instance to a job defined in another.
- For more information about the profile attribute, see the *Reference Guide*. You can also use CA WCC to assign a profile to one or more jobs.

Convert Job Profiles to the New Format (Windows Only)

In Unicenter AutoSys JM r4.5 and r11 on Windows, job profile information was stored in the Windows registry. To upgrade to r11.3, the job profile information in the registry must be converted to a file format that is compatible with the new CA Workload Automation Agent for Windows. When you upgrade CA Workload Automation AE, the upgrade process automatically converts the job profiles. You can also issue the autoprofm command to manually convert profiles.

To convert job profiles to the new format

1. Click Start, Programs, CA, Workload Automation AE, Command Prompt (*instance_name*).

   The CA Workload Automation AE command prompt opens. The command prompt presets all the environment variables for the instance.

2. Enter the following command:

   ```
   autoprofm -P directory [-N agent_name] [-x] [-?]
   ```

   The job profiles in the Windows registry are converted to the new format and stored in text files. The new files are stored in the directory specified by the -P option.

   When a job is submitted in r11.3, the agent refers to the converted profile specified in the profile attribute and sources the environment variables.

Note: For more information about the autoprofm command, see the *Reference Guide*.

Must Start Times and Must Complete Times

You can define the time that a job must start or complete by. If the job does not start by its must start time or complete by its must complete time, an alert is issued. Defining must start times and must complete times is helpful when you want to be notified that a job has not started or completed on time. For example, a must start alarm can alert you to investigate whether the job's starting conditions have not been satisfied or whether the job is stuck in the STARTING state.

You can specify more than one must start time or must complete time for a job. If the job has multiple start times, you must specify the same number of must start times or must complete times. For example, if the job runs at three different times during the day, you must specify three must start times, corresponding to each run of the job.

To define must start times, add the must_start_times attribute to your job definition.

To define must complete times, add the must_complete_times attribute to your job definition.

You can specify both must_start_times and must_complete_times attributes in the same job definition.

After the job is defined, you can issue the autorep -q command to display the must start times and the must complete times. You can issue the autorep -d command to view the alarms generated. You can also view the scheduler log file (event_demon.$AUTOSERV on UNIX and event_demon.%AUTOSERV% on Windows) to see which alarms were issued.

Note: For more information about the syntax for the must_start_times and must_complete_times attributes, see the *Reference Guide*.

How Must Start Times and Must Complete Times Work

When you define a job with the must_start_times or must_complete_times attribute, the job definition and corresponding events are added to the database. The scheduler checks the events to determine whether to issue alarms.

CA Workload Automation AE uses the following process to track the must start and must complete times:

- The job is inserted in the event table in the database.
- If the job definition includes the must_start_times attribute, a CHK_START event corresponding to the next must start time is inserted in the event table.
- If the job definition includes the must_complete_times attribute, the CHK_COMPLETE event corresponding to the next must complete time is inserted in the event table.

- The scheduler checks the CHK_START event to see whether the job has started successfully.
- If the job has not started, a MUST_START_ALARM is issued. An alert is written to the scheduler log file to indicate that the CHK_START criteria has not been satisfied.
- The scheduler checks the CHK_COMPLETE event to see whether the job has completed successfully. The scheduler checks for the SUCCESS, FAILURE, or TERMINATED events.
- If the job has not completed, a MUST_COMPLETE_ALARM is issued. An alert is written to the scheduler log file to indicate that the CHK_COMPLETE criteria has not been satisfied.
- After the job completes, the scheduler calculates the next must start time and must complete time for the job and inserts the following events in the event table:
  - A new STARTJOB event
  - A new CHK_START event for the next must start time
  - A new CHK_COMPLETE event for the next must complete time

Examples: Specifying Must Start Times and Must Complete Times

The following examples are jobs that have must start times and must complete times defined:

Example: Specify Absolute Must Start and Must Complete Times

This example defines a job to run every day at 10:00 a.m., 11:00 a.m., and 12:00 p.m. The job must start by 10:02 a.m., 11:02 a.m., and 12:02 p.m. respectively. The job must complete by 10:08 a.m., 11:08 a.m., and 12:08 p.m. respectively. Otherwise, an alarm is issued for each missed start or complete time.

```
insert_job: test_must_start_complete
command: /opt/StartTransactions.sh
machine: localhost
date_conditions: y
days_of_week: all
start_times: "10:00, 11:00, 12:00"
must_start_times: "10:02, 11:02, 12:02"
must_complete_times: "10:08, 11:08, 12:08"
```

Note: The number of must start times and must complete times must match the number of start times. Otherwise, the job cannot be defined. For example, the job cannot be defined if it has one start time and two must start and complete times, as follows:

```
start_times: "10:00"
must_start_times: "10:02, 12:02"
must_complete_times: "10:08, 12:08"
```

Example: Specify Absolute Must Start and Must Complete Times on the Next Day

This example defines a job to run every day at 12:00 a.m. Suppose that you want each job run to start by 10:10 a.m. the next day and end by 10:12 a.m. the next day. You must specify 34:10 in the must_start_times attribute and 34:12 in the must_complete_times attribute.

```
insert_job: job3
command: echo "hello"
machine: localhost
date_conditions: y
days_of_week: all
start_times: "12:00"
must_start_times: "34:10"
must_complete_times: "34:12"
```

The must start time is calculated as follows:

```
must start time + 24 hours
= 10:10 + 24 hours
= 34:10
```

The must complete time is calculated as follows:

```
must complete time + 24 hours
= 10:12 + 24 hours
= 34:12
```

If a job run does not start by the must start time, a MUST_START_ALARM is issued to notify you that the job has not started on time. If a job run does not complete by the must complete time, a MUST_COMPLETE_ALARM is issued to notify you that the job has not completed on time.

Example: Specify Relative Must Start and Must Complete Times

This example defines a job to run every day at 10:00 a.m., 11:00 a.m., and 12:00 p.m. Each job run must start within 3 minutes after each start time (10:03 a.m., 11:03 a.m., and 12:03 p.m.). Each job run must complete within 8 minutes after each start time (10:08 a.m., 11:08 a.m., and 12:08 p.m.). Otherwise, an alarm is issued for each missed start or complete time.

```
insert_job: test_must_start_complete
job_type: CMD
machine: localhost
command: /opt/StartTransactions.sh
date_conditions: y
days_of_week: all
start_times: "10:00, 11:00, 12:00"
must_start_times: +3
must_complete_times: +8
```

Example: Specify Relative Must Start and Must Complete Times With start_mins

This example defines a job to run every day at 10 minute intervals every hour (for example, 2:00 p.m., 2:10 p.m., 2:20 p.m., and so on). Each job run must start within 2 minutes after the specified start times and complete within 7 minutes after the specified start times. Otherwise, an alarm is issued for each missed start or complete time. For instance, the 2:10 p.m. job run must start by 2:12 p.m. and must complete by 2:17 p.m.

```
insert_job: test_must_start_complete
job_type: CMD
machine: localhost
command: /opt/StartTransactions.sh
date_conditions: y
days_of_week: all
start_mins: 10, 20, 30, 40, 50, 00
must_start_times: +2
must_complete_times: +7
```

Delete Obsolete Job Versions

When you update or delete a job definition, previous versions of the definitions are stored in the database. To prevent the database from being overloaded with job versions, you can delete obsolete job versions. Job versions are obsolete when the job is inactive and the database no longer refers to it.

Note: We recommend that you issue the archive_events command before issuing the archive_jobs command. We also recommend that you run archive_jobs as part of your usual database maintenance.

To delete obsolete job versions

1. Do *one* of the following:
   - On UNIX, run the shell that is sourced to use CA Workload Automation AE.

     The UNIX operating system prompt is displayed.
   - On Windows, click Start, Programs, CA, Workload Automation AE, Command Prompt (*instance_name*).

     The CA Workload Automation AE command prompt opens. The command prompt presets all the environment variables for the instance.
2. Enter the following command:

   ```
   archive_jobs -j number_of_days [-d "directory_name"] [-A] [-x] [-?]
   ```

   The obsolete job versions are deleted.

Note: For more information about the archive_jobs command, see the *Reference Guide*.

Example: Delete Obsolete Job Versions

- This example deletes obsolete job versions older than 7 days:

  ```
  archive_jobs -j 7
  ```

- This example deletes obsolete job versions older than 7 days and creates the archive flat file in the $AUTOUSER/archive file (the default):

  ```
  archive_jobs -j 7 -A
  ```

- This example deletes obsolete job versions older than 7 days and creates the archive flat file in the /tmp/archive directory:

  ```
  archive_jobs -j 7 -A -d "/tmp/archive"
  ```

Chapter 5: Application Services Jobs

This section contains the following topics:

Application Services Jobs

Application Services jobs let you manage entity beans, session beans, and MBeans, publish and consume JMS messages, invoke programs over HTTP, and run other types of Java-based workload.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

You can define the following Application Services jobs:

Entity Bean

Lets you create an entity bean, update the property values of an existing entity bean, or remove an entity bean from the database.

HTTP

Lets you invoke a program over HTTP or HTTPS in a similar way to a web browser. For example, you can use the HTTP job to invoke a CGI script, a Perl script, or a servlet. The HTTP job sends a URL over HTTP using the GET method or a form over HTTP using the POST method.

JMS Publish

Lets you send a message to a queue or publish a message to a topic on a JMS server.

JMS Subscribe

Lets you consume messages from a queue or topic on a JMS server.

JMX-MBean Attribute Get

Lets you query a JMX server for the value of an MBean attribute. The returned value is stored on the computer where the Application Services agent plug-in resides.

JMX-MBean Attribute Set

Lets you change the value of an MBean attribute on a JMX server.

JMX-MBean Create Instance

Lets you create an MBean on a JMX server.

JMX-MBean Operation

Lets you invoke an operation on an MBean on a JMX server.

JMX-MBean Remove Instance

Lets you remove an MBean from a JMX server.

JMX-MBean Subscribe

Lets you monitor an MBean for a single notification or monitor continuously for notifications.

POJO

Lets you instantiate a class to create a Java object and invoke a method on it. The job is restricted to classes that take constructors with no arguments (default constructors). You can use the POJO job to invoke custom Java code on a local computer.

RMI

Lets you set up interaction between Java objects on different computers in a distributed network. Using an RMI job, you can access a remote server and invoke a method on a Java object.

Session Bean

Lets you access a session bean on an application server. This job type can make a Remote Procedure Call (RPC) to the session bean, invoke a method that defines the business logic, pass parameters to the method, and have the results returned as serialized Java output. You can access stateless and stateful session beans using the Session Bean job.

Payload Producing and Payload Consuming Jobs

A payload producing job is a job that produces binary output that is persisted as a serialized Java object.

The following job types are payload producing jobs:

- JMS Subscribe

  Note: The appservices.jms.subscribe.persist parameter must be set to true in the agent's agentparm.txt file for JMS Subscribe jobs to be payload producing jobs.

- JMX-MBean Attribute Get
- JMX-MBean Attribute Set
- JMX-MBean Operation
- POJO
- RMI
- Session Bean
- SNMP Value Get
- SNMP Value Set
- Web Service

By default, the serialized Java object is stored on the agent computer in the spool directory, using the job name and a numeric suffix as the file name. You can redirect the output to a destination file.

A payload consuming job is a job that uses the output from a payload producing job as a parameter's input value.

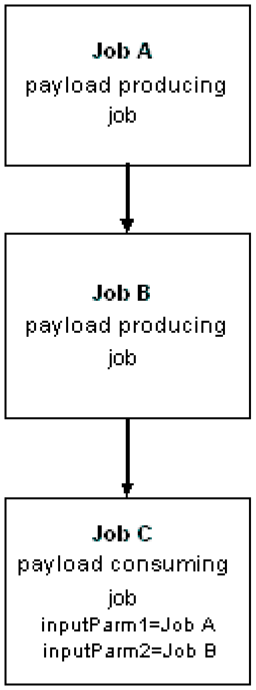
The following job types are payload consuming jobs:

- Entity Bean
- JMS Publish
- JMX-MBean Attribute Set
- JMX-MBean Create Instance
- JMX-MBean Operation
- POJO
- RMI
- Session Bean
- SNMP Value Set
- Web Service

We recommend the payload producing job be a predecessor job to the payload consuming job although it does not need to be an immediate predecessor.

Example: Pass Payload Producing Job Output as Input to Another Job

The following diagram shows the relationship between three jobs in a job flow. Job A and Job B are payload producing jobs that produce binary output. Job C is a payload consuming job that takes two parameters, inputParm1 and inputParm2. Job C uses the output from Job A and Job B as input values. In the definition of Job C, the value of inputParm1 is specified as Job A and the value of inputParm2 is specified as Job B.



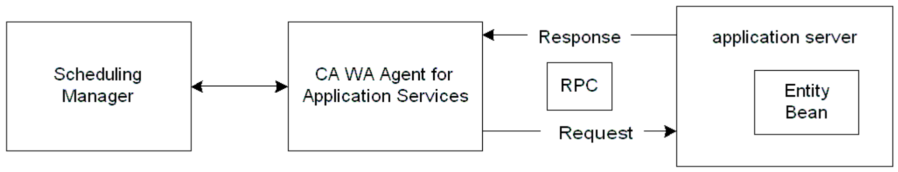
Entity Bean Jobs

An entity bean represents a data object, such as a customer, an order, or a product. Entity beans may be stored in a relational database, where each instance of the bean corresponds to a row in a database table. Each entity bean has a unique identifier known as a primary key, which is used to find a specific instance of the bean within the database. For example, a customer entity bean may use the customer number as its primary key.

Unlike session beans, which are destroyed after use, entity beans are persistent. You can use an entity bean under the following conditions:

- The bean represents a business entity, not a procedure. For example, you use an entity bean to represent an order and use a session bean to represent the procedure to process the order.
- The state of the bean must be stored. For example, if the bean instance terminates or the application server shuts down, the bean's state will still exist in a database.

The following diagram shows the functional relationship between the scheduling manager, CA WA Agent for Application Services, and an entity bean residing on an application server:



The Entity Bean job lets you create an entity bean, update the property values of an existing entity bean, or remove an entity bean from the database. To find the entity bean, the agent uses the bean's Java Naming and Directory Interface (JNDI) name along with its finder method.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define an Entity Bean job, you require the following information:

- Initial context factory supplied by the JNDI service provider
- Service provider URL for accessing the JNDI services
- Entity bean JNDI name
- Operation type (CREATE, UPDATE, or REMOVE)
- Finder method name (UPDATE and REMOVE operation types only)

Define an Entity Bean Job

You can define an Entity Bean (ENTYBEAN) job to create an entity bean, update the property values of an existing entity bean, or remove an entity bean from the database.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define an Entity Bean job

1. Insert a job and specify the following attributes in the definition:

   job_type: ENTYBEAN

   Specifies that the job type is Entity Bean.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   bean_name

   Specifies the JNDI name of the entity bean.

   initial_context_factory

   Specifies the initial context factory to use when creating the initial context. The initial context is required within the Java Naming and Directory Interface (JNDI) framework. The initial context factory is supplied by a specific provider of the naming and directory service. The factory acquires an arbitrary initial context that the application can use to connect to the application server.

   operation_type

   Specifies the operation to perform on the entity bean: CREATE, UPDATE, REMOVE.

   provider_url

   Specifies the JNDI service provider URL.

2. Specify the following attributes if the operation_type attribute is set to CREATE:

   create_name

   (Optional) Specifies the name of the create method.

   create_parameter

   (Optional) Specifies create parameters to create an entity bean in a relational database on your application server.

3. Specify the following attributes if the operation_type attribute is set to UPDATE:

   finder_name

   Specifies the name of the finder method.

   finder_parameter

   Specifies the finder parameters.

   method_name

   Specifies the method to be invoked on the application server.

   modify_parameter

   (Optional) Specifies the modify parameters.

4. Specify the following attributes if the operation_type attribute is set to REMOVE:

   finder_name

   Specifies the name of the finder method.

   finder_parameter

   (Optional) Specifies the finder parameters.

5. (Optional) Specify optional Entity Bean attributes:

   - j2ee_user
   - job_class

6. (Optional) Specify common attributes that apply to all job types.

   The Entity Bean job is defined. When the job runs, it creates an entity bean, updates the property values of an existing entity bean, or removes an entity bean from the database.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Create an Entity Bean

Suppose that you want to create an entity bean that stores information about a customer such as the customer ID and phone number. The initial context factory supplied by the JNDI service provider is weblogic.jndi.WLInitialContextFactory. The service provider's URL is t3://localhost:7001, where localhost is the domain name of the WebLogic application server and 7001 is the ORB port. When the job runs, the entity bean instance is created.

```
insert_job: create
job_type: ENTYBEAN
machine: appagent
initial_context_factory: weblogic.jndi.WLInitialContextFactory
provider_url: "t3://localhost:7001"
bean_name: customer
create_name: createcustomer
operation_type: CREATE
create_parameter: String="customerid", String="800-555-0100"
```

Example: Update an Entity Bean

Suppose that you want to update the phone number for the Acme company to 800-555-0199. The customer entity bean stores the customer ID and phone number. The primary key for the customer is the customer ID. To find the entity bean, the job uses the Acme's customer ID. When the job runs, the Acme company's phone number is changed.

```
insert_job: update
job_type: ENTYBEAN
machine: appagent
initial_context_factory: weblogic.jndi.WLInitialContextFactory
provider_url: "t3://localhost:7001"
bean_name: customer
operation_type: UPDATE
method_name: changephone
finder_name: findByPrimaryKey
finder_parameter: String="customerid"
modify_parameter: String="800-555-0199"
```

Example: Remove an Entity Bean

Suppose that you want to remove the customer record for the Acme customer. The record is stored in the database by the customer ID. When the job runs, the row in the customer table that corresponds to the Acme customer ID is removed.

```
insert_job: remove
job_type: ENTYBEAN
machine: appagent
initial_context_factory: weblogic.jndi.WLInitialContextFactory
provider_url: "t3://localhost:7001"
bean_name: customer
operation_type: REMOVE
finder_name: findByPrimaryKey
finder_parameter: String="customerid"
```

More information:

Insert a Job (see page 87)

HTTP Jobs

The HTTP job invokes a program over HTTP in a similar way to a web browser. For example, you can use the HTTP job to invoke a CGI script, a Perl script, or a servlet. The HTTP job sends a URL over HTTP using the GET method or a form over HTTP using the POST method. The output of the invocation is returned in the job's spool file.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

The GET method requests data and sends the data as part of the URL. The POST method submits data and is the preferred method for sending lengthy form data.

To define an HTTP job, you require the following information:

- URL of the application server
- Program or servlet to invoke

Note: If your company has a firewall and you must communicate through a proxy server to access a computer outside the firewall, agent configuration is required. For more information on configuring the agent for a proxy, see the *CA Workload Automation Agent for Application Services Implementation Guide*.

Define an HTTP Job

You can define an HTTP job to invoke a program over HTTP.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define an HTTP job

1. Insert a job and specify the following attributes in the definition:

   job_type: HTTP

   Specifies that the job type is HTTP.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   provider_url

   Specifies the host where the program or servlet you want to invoke resides.

2. (Optional) Specify optional HTTP attributes:
   - filter
   - invocation_type
   - j2ee_authentication_order
   - j2ee_conn_domain
   - j2ee_conn_origin
   - j2ee_no_global_proxy_defaults
   - j2ee_parameter
   - j2ee_proxy_domain
   - j2ee_proxy_host
   - j2ee_proxy_origin_host
   - j2ee_proxy_port
   - j2ee_proxy_user
   - job_class
   - method_name

3. (Optional) Specify common attributes that apply to all job types.

   The HTTP job is defined. When the job runs, it invokes a program over HTTP.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Define an HTTP Job to Perform a Google Search

Suppose that you want to define a job to perform a Google search and have the results returned to the job's spool file. You also want to refine your search results by specifying a filter. In this example, the job uses the HTTP GET method to perform the Google search on "ca workload automation". When the job runs, the job's spool file includes all matches that contain the filter AE.

```
insert_job: google
job_type: HTTP
machine: appagent
invocation_type: GET
provider_url: "http://google.com/search"
j2ee_authentication_order: BASIC,DIGEST,NTLM
filter: .*AE.*
j2ee_parameter: q="ca workload automation"
```

Example: Define an HTTP Job to Subscribe to a Mailing List

Suppose that you want to define a job to subscribe to a mailing list located on a local server. You want to add the email address test@abc.com to the list. The servlet path is /examples/servlets/servlet/TheServlet.

```
insert_job: subscribe
job_type: HTTP
machine: appagent
invocation_type: POST
provider_url: "http://localhost:8080"
method_name: /examples/servlets/servlet/TheServlet
j2ee_parameter: key1="subscribe", key2="test@abc.com"
```

More information:

Insert a Job (see page 87)

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following HTTP job attributes have default values:

invocation_type

Specifies whether to send the URL over HTTP using the GET or POST method.

Default: POST

j2ee_conn_origin

Specifies the domain for NTLM connection authentication.

Default: The computer name where the agent is running

j2ee_no_global_proxy_defaults

Specifies whether to use the global proxy configuration specified by the proxy parameters in the agentparm.txt file.

Default: Y (The job does not use the global proxy configuration specified by the proxy parameters in the agentparm.txt file.)

j2ee_proxy_domain

Specifies the domain for proxy authentication.

Default: http.proxyDomain agent parameter, if specified

j2ee_proxy_host

Specifies the proxy host name to use for the request.

Default: http.proxyHost agent parameter, if specified

j2ee_proxy_origin_host

Specifies the origin host name for proxy authentication.

Default: The computer name where the agent is running

j2ee_proxy_port

Specifies the proxy port to use for the request.

Default: 80

j2ee_proxy_user

Specifies the user name required for proxy authentication.

Default: http.proxyUser agent parameter, if specified

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Define an HTTP Job to Send a URL Over HTTP

Several attributes in the following job definition override the default values.

This example performs an HTTP query using the HTTP GET method. The output of the invocation is returned in the job's spool file. In this example, the job specifies the connection domain and origin for NTLM authentication, overrides the global proxy defaults specified in the agentparm.txt file, and specifies the BASIC, DIGEST, and NTLM protocols for web server authentication.

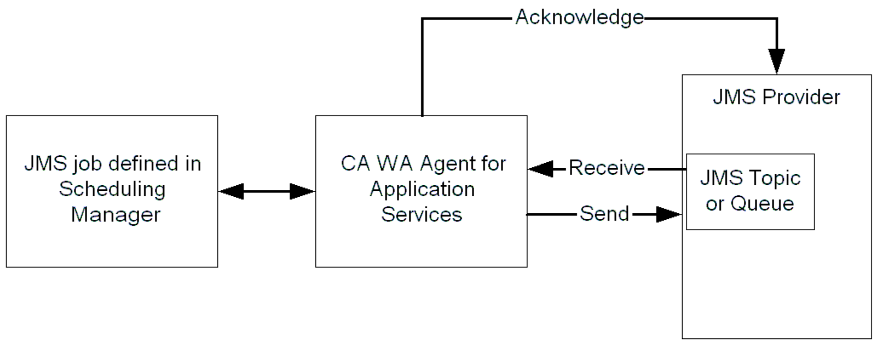
```
insert_job: HTTP.CON_USER
job_type: HTTP
machine: appagent
invocation_type: GET
provider_url: "http://host.example.com/protected"
j2ee_conn_origin: host.example.com
j2ee_conn_domain: windows_domain
j2ee_no_global_proxy_defaults: Y
j2ee_authentication_order: BASIC,DIGEST,NTLM
j2ee_proxy_domain: "http://host.domain.proxy"
j2ee_proxy_host: proxy.example.com
j2ee_proxy_origin_host: "http://host.origin.proxy"
j2ee_proxy_port: 90
j2ee_proxy_user: user01
```

JMS Publish and JMS Subscribe Jobs

Java Message Service (JMS) is the standard for enterprise messaging that lets a Java program or component (JMS client) produce and consume messages. Messages are the objects that communicate information between JMS clients.

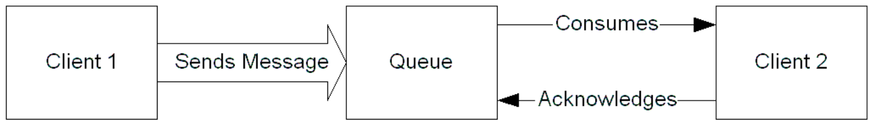
In a JMS system, a messaging server known as the JMS provider acts between two JMS clients (the publisher and the subscriber). Publishers send messages to the JMS provider while subscribers receive messages from the JMS provider.

The following diagram shows the functional relationship between the scheduling manager, the CA WA Agent for Application Services, and a JMS provider:


Acknowledge
JMS Provider
JMS job defined in Scheduling Manager
CA WA Agent for Application Services
Receive
Send
JMS Topic or Queue

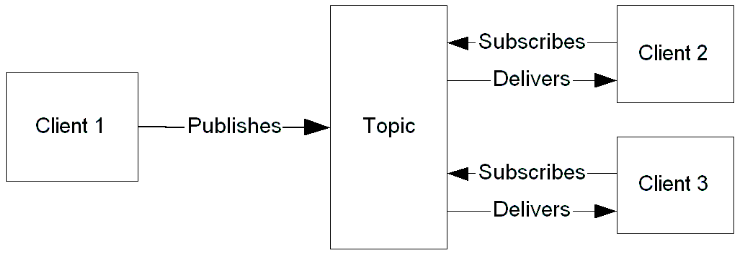
A queue is an object on the JMS server that holds messages sent by a client that are waiting to be consumed by another client. The queue retains a message until the message is consumed or the message expires.

The following diagram shows Client 2 (the subscriber) consuming a message that Client 1 (the publisher) sends to a queue:



A topic is an object a client uses to specify the target of the messages it produces and the source of the messages it consumes. A client acquires a reference to a topic on a JMS server, and sends messages to that topic. When messages arrive for that topic, the JMS provider is responsible for notifying all clients.

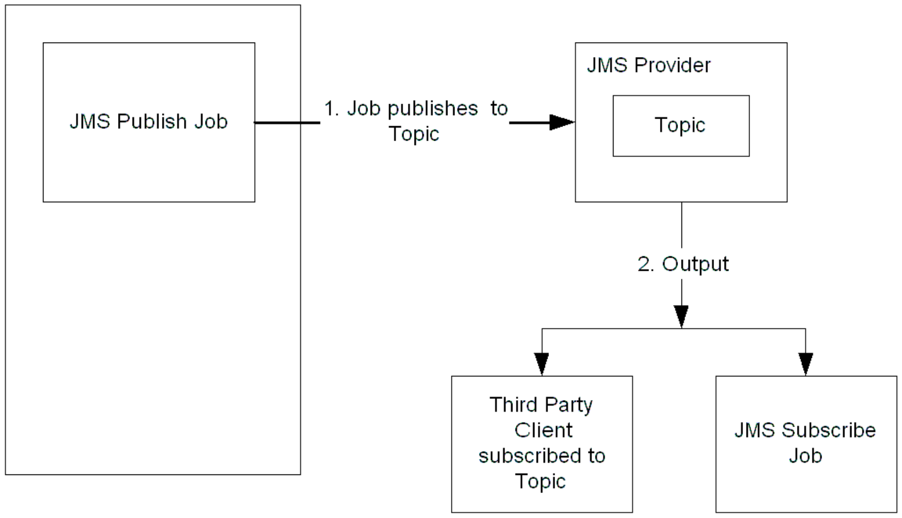
The following diagram shows two subscribers, Client 2 and Client 3, subscribed to a topic that the publisher, Client 1, publishes to:



A JMS Publish job lets you send a message to a queue or publish a message to a topic. Using a JMS Publish job to publish to a topic, you can broadcast a message to any topic subscriber. A third-party client can consume this message, or a JMS Subscribe job can listen for a particular message (using a filter).

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

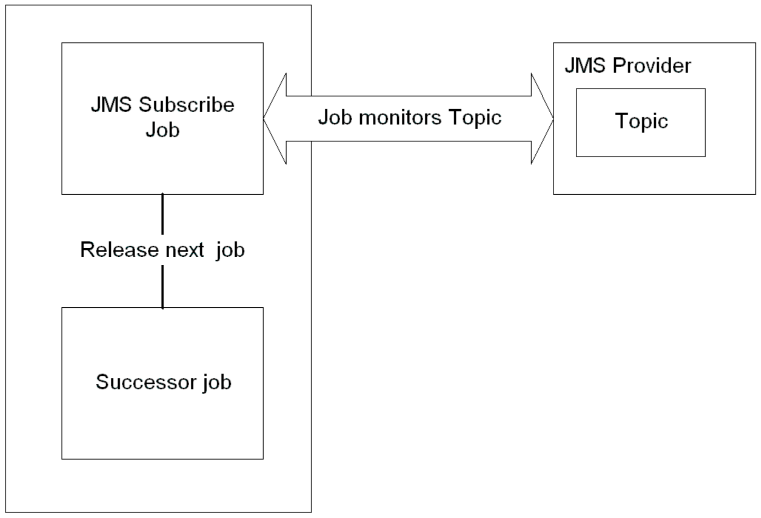
The following diagram shows a JMS Publish job scenario:


JMS Publish Job
1. Job publishes to Topic
JMS Provider
Topic
2. Output
Third Party Client subscribed to Topic
JMS Subscribe Job

A JMS Subscribe job lets you consume messages from a queue or topic. Using a filter that you define within the job definition, the agent monitors the topic or queue output for specific data. The scheduling manager then sends the message that meets the filter criteria to a destination file you specify. You can define the job to continuously monitor JMS messages.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

The following diagram shows a JMS Subscribe job scenario:



To define a JMS Publish or JMS Subscribe job, you require the following information:

- Initial context factory supplied by the Java Naming and Directory Interface (JNDI) service provider
- JMS provider URL for accessing the JNDI services
- Connection factory JNDI name that looks up the referenced topic or queue
- JNDI name of the topic or queue on the JMS server
- Java class of the JMS message to send or publish

Define a JMS Publish Job

You can define a JMS Publish job to send a message to a queue or publish a message to a topic.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define a JMS Publish job

1. Insert a job and specify the following attributes in the definition:

   job_type: JMSPUB

   Specifies that the job type is JMS Publish.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   connection_factory

   Specifies the connection factory JNDI name. The connection factory contains all the bindings needed to look up the referenced topic or queue. JMS jobs use the connection factory to create a connection with the JMS provider.

   destination_name

   Specifies the JNDI name of the topic or queue. The job uses the JNDI name to indicate the destination where messages are received.

   initial_context_factory

   Specifies the initial context factory to use when creating the initial context. The initial context is required within the Java Naming and Directory Interface (JNDI) framework. The initial context factory is supplied by a specific provider of the naming and directory service. The factory acquires an arbitrary initial context that the application can use to connect to the application server.

   j2ee_parameter

   Specifies the message to send to a queue or publish to a topic.

message_class

Specifies the Java class of the JMS message.

provider_url

Specifies the JNDI service provider URL.

2. (Optional) Specify optional JMS Publish attributes:
   - j2ee_user
   - job_class
   - use_topic
3. (Optional) Specify common attributes that apply to all job types.

   The JMS Publish job is defined. When the job runs, it sends a message to a queue or publishes a message to a topic.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Publish a Message to the WebSphere Application Server

This example publishes the message "this is my message" to the queue named Queue. The Java class of the message is String. The initial context factory supplied by the JNDI service provider is com.ibm.websphere.naming.WsnInitialContextFactory. The service provider's URL is iiop://172.24.0.0:2809, where 172.24.0.0 is the IP address of the WebSphere MQ server and 2809 is the ORB port. The job uses the cyberuser JNDI user name to gain access to the connection factory named ConnectionFactory.

```
insert_job: publish
job_type: JMSPUB
machine: appagent
initial_context_factory: com.ibm.websphere.naming.WsnInitialContextFactory
provider_url: "iiop://172.24.0.0:2809"
connection_factory: ConnectionFactory
destination_name: Queue
use_topic: FALSE
message_class: String
j2ee_user: cyberuser
j2ee_parameter: java.lang.String="this is my message"
```

Note: The agent does not support JMS messaging on IBM WebSphere. If you have IBM WebSphere MQ, your agent administrator can set up the agent plug-in to run JMS Publish and JMS Subscribe for JMS queues. JMS topics are not supported on IBM WebSphere MQ.

More information:

Insert a Job (see page 87)

Define a JMS Subscribe Job

You can define a JMS Subscribe job to consume messages from a queue or topic.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define a JMS Subscribe job

1. Insert a job and specify the following attributes in the definition:

   job_type: JMSSUB

   Specifies that the job type is JMS Subscribe.

   machine

   Specifies the name of an existing machine definition where the agent runs.

connection_factory

Specifies the connection factory JNDI name. The connection factory contains all the bindings needed to look up the referenced topic or queue. JMS jobs use the connection factory to create a connection with the JMS provider.

destination_name

Specifies the JNDI name of the topic or queue. The job uses the JNDI name to indicate the destination where messages are received.

initial_context_factory

Specifies the initial context factory to use when creating the initial context. The initial context is required within the Java Naming and Directory Interface (JNDI) framework. The initial context factory is supplied by a specific provider of the naming and directory service. The factory acquires an arbitrary initial context that the application can use to connect to the application server.

provider_url

Specifies the JNDI service provider URL.

2. (Optional) Specify optional JMS Subscribe attributes:
   - continuous
   - destination_file
   - filter
   - j2ee_user
   - job_class
   - job_terminator
   - use_topic
3. (Optional) Specify common attributes that apply to all job types.

   The JMS Subscribe job is defined. When the job runs, it consumes messages from a queue or topic.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor a Queue on a WebLogic Application Server

This example continuously monitors the queue named Queue (residing on WebLogic) for a message matching the filter criteria. The consumed messages from the queue are stored in the file /export/home/user1/outputfile1. The service provider's URL is t3://172.24.0.0:7001, where 172.24.0.0 is the IP address of the WebLogic Application server and 7001 is the ORB port.

```
insert_job: monitor
job_type: JMSSUB
machine: appagent
initial_context_factory: weblogic.jndi.WLInitialContextFactory
provider_url: "t3://172.24.0.0:7001"
connection_factory: ConnectionFactory
destination_name: Queue
continuous: Y
filter: abc\s...\s[a-zA-Z]+\sFilter![\sa-z0-9]+
use_topic: FALSE
destination_file: /export/home/user1/outputfile1
j2ee_user: cyberuser
```

In this example, the regular expression used as the filter criteria can be defined as follows:

```
abc\s...\s[a-zA-Z]+\sFilter![\sa-z0-9]+
```

abc\s

Specifies the text abc, followed by white space.

...\s

Specifies any three characters, followed by white space.

[a-zA-Z]+\s

Specifies at least one letter, followed by white space.

Filter![\sa-z0-9]+

Specifies the text Filter!, followed by at least one of the following: white space or digit or lower case letter.

Example: abc vvv B Filter! 95

More information:

Insert a Job (see page 87)

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following JMS job attributes have default values:

continuous (JMS Subscribe jobs only)

Specifies whether the job monitors the topic or queue continuously for messages.

Default: N (The job immediately checks for the condition and completes.)

job_terminator (JMS Subscribe jobs only)

Specifies whether to terminate the job if its containing box fails or terminates.

Default: n (The job does not terminate if its containing box fails or terminates.)

destination_file (JMS Subscribe jobs only)

Specifies the output destination file for the consumed messages.

Default: spooldir agent parameter, if specified

use_topic

Specifies whether to send or publish messages to a topic or queue.

Default: FALSE (The job sends or publishes messages to a queue.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Specify Optional Attributes in a JMS Subscribe Job

The continuous and destination_file attributes in the following job definition override the default values.

This example continuously monitors the queue named Queue (residing on WebLogic) for a message matching the filter criteria. The consumed messages from the queue are stored in the file /export/home/user1/outputfile1. The service provider's URL is t3://172.24.0.0:7001, where 172.24.0.0 is the IP address of the WebLogic Application server and 7001 is the ORB port.

```
insert_job: monitor
job_type: JMSSUB
machine: appagent
initial_context_factory: weblogic.jndi.WLInitialContextFactory
provider_url: "t3://172.24.0.0:7001"
connection_factory: ConnectionFactory
destination_name: Queue
continuous: Y
filter: abc\s...\s[a-zA-Z]+\sFilter![\sa-z0-9]+
use_topic: FALSE
destination_file: /export/home/user1/outputfile1
j2ee_user: cyberuser
```

In this example, the regular expression used as the filter criteria can be defined as follows:

```
abc\s...\s[a-zA-Z]+\sFilter![\sa-z0-9]+
```

abc\s

Specifies the text abc, followed by white space.

...\s

Specifies any three characters, followed by white space.

[a-zA-Z]+\s

Specifies at least one letter, followed by white space.

Filter![\sa-z0-9]+

Specifies the text Filter!, followed by at least one of the following: white space or digit or lower case letter.

Example: abc vvv B Filter! 95

JMX Jobs

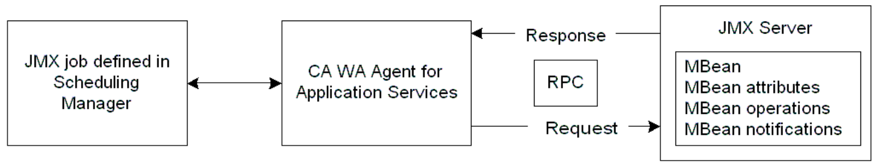
Java Management Extension (JMX) technology is included in the Java Standard Edition (SE) platform, version 5 and higher. JMX lets you remotely access applications, using a Remote Method Invocation (RMI) connector, for monitoring and management purposes.

JMX jobs let you access a remote JMX server that advertises MBeans. An MBean is a managed bean (Java object) that represents an application, a device, or any resource that you want to manage. An MBean contains a set of attributes and a set of operations that can be invoked. Some MBeans can send out notifications, for example, when an attribute changes.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

Consider an MBean named Config that represents an application's configuration. The configuration parameters within that application are represented in Config by a set of attributes. Getting the attribute named cachesize, for example, returns the current value of the cachesize. Setting the value updates the cachesize. The Config MBean can send out a notification every time the cachesize changes. An operation named update, for example, can save changes to the configuration parameters.

The following diagram shows the functional relationship between the scheduling manager, CA WA Agent for Application Services, and the JMX server:



The JMX jobs provide support for getting and setting JMX MBean attributes, invoking JMX MBean operations, subscribing to MBean notifications, and creating and removing instances of MBeans on a JMX server.

You can define the following six types of JMX jobs:

- JMX-MBean Attribute Get
- JMX-MBean Attribute Set
- JMX-MBean Create Instance
- JMX-MBean Operation
- JMX-MBean Remove Instance
- JMX-MBean Subscribe

The JMX-MBean Attribute Set, JMX-MBean Create Instance, and JMX-MBean Operation jobs support calls to MBeans that can involve passing parameters. Each parameter can be an actual value or a serialized Java object passed by another job. When the JMX-MBean Operation job invokes an operation on an MBean that passes parameters, the parameters are passed to the MBean and the returned serialized Java object is stored on the agent computer in the spool directory or in a destination file you specify.

To define JMX jobs, you require a URL to connect to the JMX server using an RMI connector.

Define a JMX-MBean Attribute Get Job

You can define a JMX-MBean Attribute Get job to query a JMX server for the value of an MBean attribute. The returned value is stored on the computer where the agent resides. You can specify a success pattern to determine the job's success or failure. If the returned attribute value matches the success pattern, the job completes successfully; otherwise, it fails.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define a JMX-MBean Attribute Get job

1. Insert a job and specify the following attributes in the definition:

   job_type: JMXMAG

   Specifies that the job type is JMX-MBean Attribute Get.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   mbean_attr

   Specifies the name of the MBean attribute that you want to query.

   mbean_name

   Specifies the full object name of an MBean.

   URL

   Specifies the URL to connect to the JMX server using an RMI connector.

2. (Optional) Specify optional JMX-MBean Attribute Get attributes:
   - job_class
   - jmx_user
   - success_pattern
3. (Optional) Specify common attributes that apply to all job types.

   The JMX-MBean Attribute Get job is defined. When the job runs, it queries a JMX server for the value of an MBean attribute.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Query a JMX Server for the Value of an MBean Attribute

Suppose that you want to know the value of the cachesize attribute for the Config MBean. The URL for the JMX server is service:jmx:rmi:///jndi/rmi://localhost:9999/server, where localhost is the host name and 9999 is the port number.

```
insert_job: query
job_type: JMXMAG
machine: appagent
URL: "service:jmx:rmi:///jndi/rmi://localhost:9999/server"
mbean_name: "DefaultDomain:index=1,type=Config"
mbean_attr: cachesize
```

More information:

Insert a Job (see page 87)

Define a JMX-MBean Attribute Set Job

You can define a JMX-MBean Attribute Set job to change the value of an MBean attribute on a JMX server. You can specify a set value for the attribute or use the serialized Java object passed by another job. When the attribute is set, the job returns the original attribute value as output. You can specify a success pattern to determine the job's success or failure. If the job's output matches the success pattern, the job completes successfully; otherwise, it fails.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define a JMX-MBean Attribute Set job

1. Insert a job and specify the following attributes in the definition:

   job_type: JMXMAS

   Specifies that the job type is JMX-MBean Attribute Set.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   mbean_attr

   Specifies the name of the MBean attribute that you want to set.

   mbean_name

   Specifies the full object name of an MBean.

   URL

   Specifies the URL to connect to the JMX server using an RMI connector.

2. (Optional) Specify optional JMX-MBean Attribute Set attributes:
   - jmx_parameter
   - jmx_user
   - job_class
   - success_pattern

3. (Optional) Specify common attributes that apply to all job types.

   The JMX-MBean Attribute Set job is defined. When the job runs, it changes the value of an MBean attribute on a JMX server.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Change the Value of an MBean Attribute

Suppose that you want to set the value of the State attribute of the cdc.jmx.SimpleDynamic MBean to the serialized Java object returned by a JMX-MBean Attribute Set job named size.

```
insert_job: change
job_type: JMXMAS
machine: appagent
URL: "service:jmx:rmi:///jndi/rmi://agenttest:5099/jmxserver"
mbean_name: "DefaultDomain:index=1,type=cdc.jmx.SimpleDynamic"
mbean_attr: State
jmx_parameter: payload_job=size
condition: success(size)
```

More information:

Insert a Job (see page 87)

Define a JMX-MBean Create Instance Job

You can define a JMX-MBean Create Instance job to create an MBean on a JMX server.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define a JMX-MBean Create Instance job

1. Insert a job and specify the following attributes in the definition:

   job_type: JMXMC

   Specifies that the job type is JMX-MBean Create Instance.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   class_name

   Specifies the fully qualified Java class of the MBean object.

   mbean_name

   Specifies the full object name of an MBean.

   URL

   Specifies the URL to connect to the JMX server using an RMI connector.

2. (Optional) Specify optional JMX-MBean Create Instance attributes:
   - jmx_parameter
   - jmx_user
   - job_class
3. (Optional) Specify common attributes that apply to all job types.

   The JMX-MBean Create Instance job is defined. When the job runs, it creates an MBean on a JMX server.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Create an MBean Instance on a JMX Server

Suppose that you want to create an MBean instance on a JMX server. The job uses the cdc.jmx.SimpleDynamic class. The constructor of the class takes a single string parameter with the value "Hello".

```
insert_job: create
job_type: JMXMC
machine: appagent
URL: "service:jmx:rmi:///jndi/rmi://agenttest:5099/jmxserver"
mbean_name: "DefaultDomain:index=CreateIns1,type=cdc.jmx.SimpleDynamic"
class_name: cdc.jmx.SimpleDynamic
jmx_parameter: java.lang.String="Hello"
```

More information:

Insert a Job (see page 87)

Define a JMX-MBean Operation Job

You can define a JMX-MBean Operation job to invoke an operation on an MBean. You can specify one or more parameter values to pass to the operation. You can specify a success pattern to determine the job's success or failure. If the operation's output matches the success pattern, the job completes successfully; otherwise, it fails.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define a JMX-MBean Operation job

1. Insert a job and specify the following attributes in the definition:

   job_type: JMXMOP

   Specifies that the job type is JMX-MBean Operation.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   mbean_name

   Specifies the full object name of an MBean.

   mbean_operation

   Specifies the operation to be invoked.

   URL

   Specifies the URL to connect to the JMX server using an RMI connector.

2. (Optional) Specify optional JMX-MBean Operation attributes:

   - jmx_parameter
   - jmx_user
   - job_class
   - success_pattern

3. (Optional) Specify common attributes that apply to all job types.

   The JMX-MBean Operation job is defined. When the job runs, it invokes an operation on an MBean.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Invoke an Operation on an MBean

Suppose that you want to invoke the resetmem operation on the config MBean to reset the value of the memory parameter to 50.

```
insert_job: reset
job_type: JMXMOP
machine: agent
URL: "service:jmx:rmi:///jndi/rmi://localhost:9999/server"
mbean_name: "DefaultDomain:index=1,type=Config"
mbean_operation: resetmem
jmx_parameter: Integer=50
```

Example: Pass Payload Producing Output as Input to Payload Consuming Job

Suppose that you want to use a JMX-MBean Operation job to invoke a method on an MBean and pass the output of the method as input to another JMX-MBean Operation job.

In this example, the first job, test_JMXMOP2a, is a payload producing job. It takes a single input parameter and invokes the reset method on the MBean. The output of this job is stored as a serialized Java object on the computer where the agent resides.

The second job, test_JMXMOP2b, is a payload consuming job. It takes two input parameters: the string "Hello" and the serialized Java object produced by the first job. The two input parameters are passed to the reset method, which is invoked on the MBean.

```
insert_job: test_JMXMOP2a
machine: localhost
job_type: JMXMOP
url: "service:jmx:rmi:///jndi/rmi://localhost:9999/server"
mbean_name: "DefaultDomain:type=SimpleStandard,index=1"
mbean_operation: reset
jmx_parameter: String="Hello"
```

```
insert_job: test_JMXMOP2b
machine: localhost
job_type: JMXMOP
url: "service:jmx:rmi:///jndi/rmi://localhost:9999/server"
mbean_name: "DefaultDomain:type=SimpleStandard,index=1"
mbean_operation: reset
jmx_parameter: String="Hello", payload_job=test_JMXMOP2a
condition: S(test_JMXMOP2a)
```

More information:

Insert a Job (see page 87)

Define a JMX-MBean Remove Instance Job

You can define a JMX-MBean Remove Instance job to remove an MBean from a JMX server.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define a JMX-MBean Remove Instance job

1. Insert a job and specify the following attributes in the definition:

   job_type: JMXMREM

   Specifies that the job type is JMX-MBean Remove Instance.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   mbean_name

   Specifies the full object name of an MBean.

   URL

   Specifies the URL to connect to the JMX server using an RMI connector.

2. (Optional) Specify optional JMX-MBean Remove Instance attributes:
   - jmx_user
   - job_class
3. (Optional) Specify common attributes that apply to all job types.

   The JMX-MBean Remove Instance job is defined. When the job runs, it removes an MBean from a JMX server.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Remove an MBean Instance from a JMX Server

Suppose that you want to remove an MBean instance.

```
insert_job: remove
job_type: JMXMREM
machine: appagent
URL: "service:jmx:rmi:///jndi/rmi://agenttest:5099/jmxserver"
mbean_name: "DefaultDomain:index=CreateIns1,type=cdc.jmx.SimpleDynamic"
```

More information:

Insert a Job (see page 87)

Define a JMX-MBean Subscribe Job

You can define a JMX-MBean Subscribe job to monitor an MBean for a single notification or monitor continuously for notifications. You can filter the notifications the job monitors by attributes or by type of notifications.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define a JMX-MBean Subscribe job

1. Insert a job and specify the following attributes in the definition:

   job_type: JMXSUB

   Specifies that the job type is JMX-MBean Subscribe.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   mbean_name

   Specifies the full object name of an MBean.

   URL

   Specifies the URL to connect to the JMX server using an RMI connector.

2. (Optional) Specify optional JMX-MBean Subscribe attributes:
   - continuous
   - filter
   - filter_type
   - jmx_user
   - job_class
   - job_terminator
3. (Optional) Specify common attributes that apply to all job types.

   The JMX-MBean Subscribe job is defined. When the job runs, it monitors an MBean for a single notification or continuously for notifications.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor for a Change to a Specific MBean Attribute

Suppose that you want to monitor for a change to the cachesize attribute of the MBean named Config. The job filters the notifications the MBean sends by attribute. When the cachesize attribute changes, the job completes.

```
insert_job: change
job_type: JMXSUB
machine: agent
URL: "service:jmx:rmi:///jndi/rmi://localhost:9999/server"
mbean_name: "DefaultDomain:index=1,type=Config"
filter_type: Attributes
filter: cachesize
```

More information:

Insert a Job (see page 87)

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following JMX Subscribe job attributes have default values:

continuous

Specifies whether the job monitors the MBean continuously for notifications.

Default: N (The job immediately checks for the condition and completes.)

filter_type

Specifies whether to filter notifications by attribute or by notification type.

Default: Attributes (The job filters notifications by attribute.)

job_terminator

Specifies whether to terminate the job if its containing box fails or terminates.

Default: n (The job does not terminate if its containing box fails or terminates.)

destination_file

Specifies the output destination file for the Java serialized object produced by the job.

Default: spooldir agent parameter, if specified

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor for Changes to Any MBean Attribute

The continuous attribute in the following job definition overrides the default value.

Suppose that you want to set up continuous monitoring for changes to any attribute of the MBean named Config. Each time an attribute changes, an alert is written to the scheduler log file.

```
insert_job: change
job_type: JMXSUB
machine: appagent
URL: "service:jmx:rmi:///jndi/rmi://localhost:9999/server"
mbean_name: "DefaultDomain:index=1,type=Config"
continuous: Y
filter_type: Types
filter: jmx.attribute.change
```

POJO Jobs

A Plain Old Java Object (POJO) is a Java object that follows the Java Language Specification only. All Java objects are POJOs.

The POJO job lets you instantiate a class to create a Java object and invoke a method on it. The job is restricted to classes that take constructors with no arguments (default constructors).

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and either CA WA Agent for Application Services or CA WA Agent for Web Services.

You can use the POJO job to invoke custom Java code on a local computer. POJO jobs support method calls that can involve passing parameters. The parameters can be actual values or a serialized Java object passed by another job. When the POJO job invokes a method on an object, the parameters, if any, are passed to the object and the returned values are stored in a Java serialized object file.

To define a POJO job, you require the class name and method you want to call on the instantiated object.

Note: If you use custom Java code, contact your agent administrator to verify the required JAR file is available in the jars subdirectory of the agent installation directory.

Define a POJO Job

You can define a POJO job to create a Java object instance with no arguments, invoke a method on the object instance, and store the method's output.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and either CA WA Agent for Application Services or CA WA Agent for Web Services.

To define a POJO job

1. Insert a job and specify the following attributes in the definition:

   job_type: POJO

   Specifies that the job type is POJO.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   class_name

   Specifies the Java class to instantiate.

   method_name

   Specifies the Java method to call on the instance of the Java object.

2. (Optional) Specify optional POJO attributes:
   - destination_file
   - j2ee_parameter
   - job_class
3. (Optional) Specify common attributes that apply to all job types.

   The POJO job is defined. When the job runs, it creates a Java object instance with no arguments, invokes a method on the object instance, and stores the method's output.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Invoke a Method on a Java Object Instance

Suppose that you want to define a POJO job that creates a Java String with value "5" and calls the parseInt method with the created Java String object as an argument. The parseInt method returns a Java Integer object.

```
insert_job: ignore
job_type: POJO
machine: appagent
class_name: java.lang.Integer
method_name: parseInt
j2ee_parameter: java.lang.String=5
```

More information:

Insert a Job (see page 87)

RMI Jobs

Remote Method Invocation (RMI) is the Java version of a Remote Procedure Call (RPC), which is a technology that lets a program request a service from another program located in another address space. That address space could be on the same computer or on a different one.

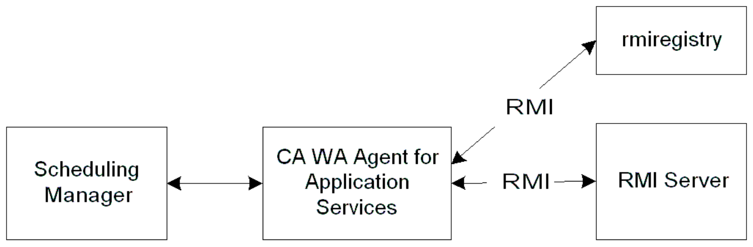
RMI jobs let you set up interaction between Java objects on different computers in a distributed network. Using an RMI job, you can access a remote server and invoke a method on a Java object. A method is a programmed procedure that is defined as a part of a Java class.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

RMI jobs support method calls to remote objects that can involve passing parameters. The parameters can be actual values or a serialized Java object passed by another job. When the RMI job invokes a method on an object that passes parameters, the parameters are passed to the remote object and the returned serialized Java object is stored on the agent computer in the spool directory or in a destination file you specify.

RMI uses a naming or directory service to locate the remote object on the remote server. To define an RMI job, you require the naming class of the Java object you want to invoke a method on. That naming class takes a name that is a java.lang.String in URL format.

The following diagram shows the functional relationship between the scheduling manager, CA WA Agent for Application Services, and an RMI Server:



Define an RMI Job

You can define an RMI job to call a method on a remote server and store the method's output.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define an RMI job

1. Insert a job and specify the following attributes in the definition:

   job_type: JAVARMI

   Specifies that the job type is RMI.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   method_name

   Specifies the method of the remote Java class to invoke.

   remote_name

   Specifies the reference location of the object you want to invoke a method on.

2. (Optional) Specify optional RMI attributes:
   - destination_file
   - j2ee_parameter
   - job_class
3. (Optional) Specify common attributes that apply to all job types.

   The RMI job is defined. When the job runs, it calls a method on a remote server and stores the method's output.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Define a Job to Start a Remote Server Immediately

Suppose that you want to invoke a method that starts a remote server using remote object activation. You want the server to start immediately.

```
insert_job: start
job_type: JAVARMI
machine: appagent
remote_name: "rmi://remotehost/Test"
method_name: startserver
j2ee_parameter: String="now"
```

More information:

Insert a Job (see page 87)

Session Bean Jobs

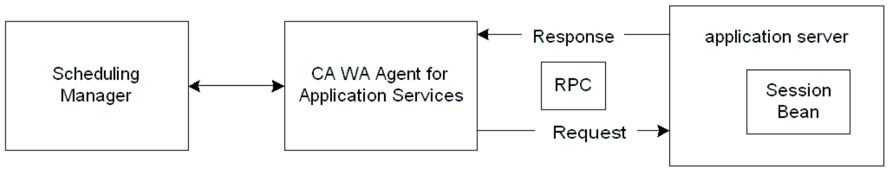
A session bean represents business logic or action to be taken (for example, charging a credit card or adding items to an online shopping cart).

Unlike entity beans, which are stored in a database, session beans may be destroyed after each use. For example, when a session bean is invoked to perform credit card validation, the application server creates an instance of that session bean, performs the business logic to validate the credit card transaction, and then destroys the session bean instance after the credit card transaction has been validated.

You can use a session bean under the following conditions:

- The bean represents a procedure and not a business entity. For example, you use a session bean to encrypt data or add items to an online shopping cart.
- The state of the bean does not have to be kept in permanent storage. For example, when the bean instance terminates or the application server shuts down, the bean's state is no longer required.

The following diagram shows the functional relationship between the scheduling manager, CA WA Agent for Application Services, and a session bean residing on an application server:



The Session Bean job lets you access a session bean on an application server. This job type can make a Remote Procedure Call (RPC) to the session bean, invoke a method that defines the business logic, pass parameters to the method, and have the results returned as serialized Java output. The output can be stored on the agent computer as text in the spool file or as a serialized Java object in the spool directory or a destination file you specify.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

You can access stateless and stateful session beans using the Session Bean job. The job acts in a similar way for both types of beans. For both stateful and stateless beans, you can specify parameters to pass to the method. When you define a stateful session bean, however, you must specify parameters to define the bean. After the method is invoked, the agent destroys the stateful bean.

Use a stateless Session Bean job to invoke a single instance of a method on the bean, such as encrypting data or sending an email to confirm an order. Use a stateful Session Bean job to invoke the same method on the bean multiple times, such as adding multiple items to an online shopping cart.

A Session Bean job requires a dedicated connection between the agent and the application server. To define a Session Bean job, you require the following information:

- Initial context factory supplied by the Java Naming and Directory Interface (JNDI) service provider
- Service provider URL for accessing the JNDI services
- Session bean JNDI name
- Method to be invoked

Define a Session Bean Job

You can define a Session Bean (SESSBEAN) job to access a stateless or stateful session bean, invoke a method on the bean, and return the results.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Application Services.

To define a Session Bean job

1. Insert a job and specify the following attributes in the definition:

   job_type: SESSBEAN

   Specifies that the job type is Session Bean.

   machine

   Specifies the name of an existing machine definition where the agent runs.

bean_name

Specifies the JNDI name of the session bean.

initial_context_factory

Specifies the initial context factory to use when creating the initial context. The initial context is required within the Java Naming and Directory Interface (JNDI) framework. The initial context factory is supplied by a specific provider of the naming and directory service. The factory acquires an arbitrary initial context that the application can use to connect to the application server.

method_name

Specifies the method to be invoked on the application server.

provider_url

Specifies the JNDI service provider URL.

2. Specify the following attributes to access a stateful session bean:

   create_method

   Specifies the name of the create method.

   create_parameter

   Specifies the create parameters.

3. (Optional) Specify optional Session Bean attributes:

   - destination_file
   - j2ee_parameter
   - j2ee_user
   - job_class

4. (Optional) Specify common attributes that apply to all job types.

   The Session Bean job is defined. When the job runs, it accesses a stateless or stateful session bean, invokes a method on the bean, and returns the results.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Invoke a Method on a Stateless Session Bean

Suppose that you want to invoke the reverse method on the CybEJBTestBean stateless session bean. The reverse method has one parameter, with type java.lang.String and value a23. The output from the reverse method is saved in the C:\Makapt15 file. The initial context factory supplied by the JNDI service provider is com.ibm.websphere.naming.WsnInitialContextFactory. The service provider's URL is iiop://172.24.0.0:2809, where 172.24.0.0 is the IP address of the WebSphere application server and 2809 is the ORB port. When the job runs, the output of the reverse method is stored in the output destination file.

```
insert_job: reverse
job_type: SESSBEAN
machine: appagent
initial_context_factory: com.ibm.websphere.naming.WsnInitialContextFactory
provider_url: "iiop://172.24.0.0:2809"
bean_name: CybEJBTestBean
method_name: reverse
destination_file: "C:\Makapt15"
j2ee_user: cyberuser
j2ee_parameter: java.lang.String="a23"
```

More information:

Insert a Job (see page 87)

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following Session Bean job attributes have default values:

create_method

Specifies the name of the create method.

Default: create

destination_file

Specifies the output destination file for the Java serialized object produced by the job.

Default: spooldir agent parameter, if specified

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Invoke a Method on a Stateful Session Bean

The create_method attribute in the following job definition overrides the default value.

Suppose that you want to access a stateful session bean for an online shopping cart. The createaddbook method creates the Shoppingcart stateful bean for the duration of the job. The addbook method adds books to the shopping cart using the book's ISBN number. In this example, the Session Bean job adds two books to the shopping cart with ISBN numbers 1551929120 and 1582701709. When the job runs, two books are added to the shopping cart.

```
insert_job: addbook
job_type: SESSBEAN
machine: appagent
initial_context_factory: com.ibm.websphere.naming.WsnInitialContextFactory
provider_url: "iiop://172.24.0.0:2809"
bean_name: Shoppingcart
create_method: createaddbook
method_name: addbook
create_parameter: String="ISBN"
j2ee_parameter: Integer[2]="1551929120,1582701709"
```

Chapter 6: Box Jobs

This section contains the following topics:

Box Jobs

A Box job (or box) is a container of other jobs. You can use it to organize and control process flow. The box itself performs no actions, although it can trigger other jobs to run. An important feature of this type of job is that boxes can contain other boxes. You can use boxes to contain other boxes that contain jobs related by starting conditions or other criteria. This feature lets you group the jobs and operate on them in a logical manner.

Box jobs are powerful tools for organizing, managing, and administering large numbers of jobs that have similar starting conditions or complex logic flows. Knowing how and when to use boxes is often the result of some experimentation.

For example, assume you want to schedule a group of jobs to start running when a File Watcher job completes successfully. Instead of making each job dependent on the File Watcher job, you can create a Box job that is dependent on the File Watcher job, remove the File Watcher job dependency from the individual jobs, and put all of those jobs in the box. When the File Watcher job completes successfully, the Box job starts, which in turn starts all the jobs it contains.

Starting Conditions for Box Jobs

When no other starting conditions are specified at the job level, a job in a box runs when the starting conditions for the box are satisfied. When several jobs in a box do not have job-level starting conditions, they all run in parallel. When any job in a box changes state, other jobs check if they are eligible to start running.

When the priority attribute is set for jobs in a box, they are processed in order of priority, highest to lowest.

Note: For more information about the priority attribute, see the *Reference Guide*.

Jobs in boxes run only once for each box execution. If you specify multiple start times for a job during one box processing cycle, only the first start time is used. This prevents jobs in boxes from inadvertently running multiple times.

CA Workload Automation AE starts a job when the current time matches, or is later than, the specified start time. In addition to explicit starting conditions, jobs in boxes have the implicit condition that the box job itself is running. This means that jobs in a box start only if the box job is running. However, if a job in a box starts and the box job is stopped, the started job runs to completion.

Note: Use caution when putting a job with more than one time-related starting condition in a box. For example, assume that a job that runs at 15 and 45 minutes past the hour is put in a box that runs at the start of every hour. The first time the box starts, the job runs at 15 minutes past the hour. A future start is then issued for 45 minutes past the hour, by which time the box has completed. As a result, the job will not run until the box runs again at the start of the next hour. At that time, the job runs as soon as the box starts because it is past its start time. The job runs, another future start job is issued for 15 minutes past the hour, the box completes, and the cycle repeats itself.

Basic Box Concepts

A *box* is a container of jobs with similar starting conditions (either date and time conditions or job dependency conditions). Use boxes to group jobs with similar scheduling parameters, not to group jobs organizationally. For example, you can group jobs that run daily at 1:00 a.m. in a box and assign them a daily start condition. However, you should not group a variety of account processing jobs with diverse starting conditions in the same box.

Default Box Job Behavior

The following default rules apply to boxes:

- Jobs in a box run only once for each box execution.
- Jobs in a box start only if the box itself has a status of RUNNING.
- Boxes are used primarily for jobs with the same starting conditions.
- A box used to group sequential jobs can contain up to 1,000 jobs.
- A box remains in RUNNING state until all the jobs it contains have run.
- A box returns a status of SUCCESS when all the jobs it contains have run and returned a status of SUCCESS.
- A box returns a status of FAILURE when all the jobs it contains have run and one or more of the jobs has returned a status of FAILURE.
- A box runs until it reaches a status of SUCCESS or FAILURE.
- Using the sendevent command to change the state of a box to INACTIVE changes the state of all the jobs it contains to INACTIVE.

More information:

Box Job Attributes and Terminators (see page 180)

Box Job Recommendations

Because all jobs in a box change status when a box starts running, you may want to use boxes to implement job cycle behavior. However, placing jobs in a box to achieve this behavior can affect your system adversely because the job status changes put a larger load on the scheduler when the box starts running.

Do not put jobs in a box solely to run reports on all of them. When you run autorep on a box, the command generates a report about the box and all the jobs it contains (unless you use the -L0 option).

Note: Job names can only contain the following characters: a-z, A-Z, 0-9, period (.), underscore (_), and hyphen (-). You cannot include spaces in a job name.

How a Box Runs

When a box starts running, the status of all the jobs it contains (including subboxes) changes to ACTIVATED, which means they are eligible to run. Because of this status change, jobs in boxes do not retain their statuses from previous box cycles.

When a box starts running, the system performs the following actions:

- Analyzes each job for additional starting conditions.
- Starts all jobs with no additional starting conditions and without any implied order or priority.
- Maintains jobs with additional starting conditions in the ACTIVATED state until those additional dependencies are met.
- Maintains the box in the RUNNING state as long as there are jobs in it with ACTIVATED or RUNNING status.
- Changes the status of the job directly from ACTIVATED to INACTIVE if its containing box is terminated before the job starts.

Note: Jobs in a box cannot start unless the box has a status of RUNNING. However, after a job starts running, it runs to completion even if the box is stopped.

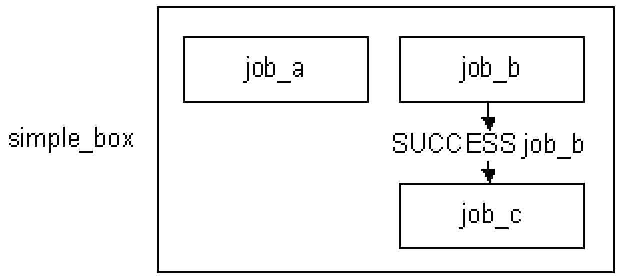
After all the jobs in a box have completed successfully, the box completes with a status of SUCCESS. The status of the box and the jobs it contains remain unchanged until the next time the box runs.

If a box changes to TERMINATED state (for example, if a user sends a KILLJOB event), it stays in TERMINATED state until the next time it is started, regardless of any later state changes of the jobs it contains.

Example: Simple Box Job

This example shows the behavior of a simple box job.

The following illustration shows a box named simple_box that contains three jobs (job_a, job_b, and job_c). job_a and job_b have no starting conditions. The starting condition for job_c is the success of job_b.



When simple_box starts running, the status of all the jobs changes to ACTIVATED. Because job_a and job_b have no additional starting conditions, they start running. When job_b completes successfully, job_c starts. When job_c completes successfully, the box completes with a SUCCESS status.

If job_b fails, job_c does not start but remains in the ACTIVATED state. Because no contingency conditions have been defined, simple_box continues running, waiting for the default completion criteria (that all jobs in the box have run) to be met.

More information:

How Job Status Changes Affect Box Status (see page 179)

How Job Status Changes Affect Box Status

If a box that is not running contains a job that changes status because of a FORCE_STARTJOB or CHANGE_STATUS event, the new job status could change the status of its containing box. A status change for the box could then trigger the start of downstream jobs that are dependent on the box.

If a box contained only one job, and the job changed status, the box status would change as shown in the following table:

Current Box Status	New Job Status	New Box Status
SUCCESS	TERMINATED or FAILURE	FAILURE
SUCCESS	SUCCESS	Box status does not change
FAILURE	SUCCESS	SUCCESS

Current Box Status	New Job Status	New Box Status
FAILURE	FAILURE	Box status does not change
INACTIVE	SUCCESS	SUCCESS
INACTIVE	TERMINATED or FAILURE	FAILURE
TERMINATED	Any change	Box status does not change

If another job is dependent on the status of the box, the status change could trigger the job to start. If the box status does not change, dependent jobs are not affected.

If the box contains other jobs in addition to the job that changed status, the status of the box is evaluated again according to the success or failure conditions assigned to the box (either the default or user-assigned). Any jobs in the box with a status of INACTIVE are ignored when the status of the box is being re-evaluated. For example, consider an INACTIVE box that contains four jobs, all with a status of INACTIVE (this is typical of a newly created box). If one of the jobs is forced to start and completes successfully, the status of the box changes to SUCCESS even though none of the other jobs ran.

Box Job Attributes and Terminators

The following sections describe how to use various job attributes to control the behavior of box jobs and the jobs they contain.

Note: For more information, see the *Reference Guide*.

Attributes in a Box Job Definition

You can use the box_success and box_failure attributes to override the default success or failure conditions for a box. When you include these attributes in a box job definition, they check what conditions must be met to put the box in a state of SUCCESS or FAILURE. When you specify success conditions for a box, you must also specify failure conditions; otherwise the product uses the default failure conditions.

Example: Non-Default Success Condition

This example shows the behavior of a simple box job for which a non-default success condition is defined.

Assume a box named simple_box that contains three jobs (job_a, job_b, and job_c), where job_a and job_b have no starting conditions and the starting condition for job_c is the successful completion of job_b. You could use the box_success attribute as follows to define a success condition for simple_box:

```
box_success: success(job_a)
```

In this case, simple_box completes successfully if job_a runs successfully, even if job_b is still running. If job_b has not completed successfully when simple_box completes, job_c changes from ACTIVATED to INACTIVE without running because the box it is in is no longer running.

Note: Do not define conflicting success and failure conditions when overriding default box terminators.

Attributes in a Job Definition

You can use the following attributes in the job definition of a job in a box to force either the job or the box to stop running:

box_terminator: y

Specifies that if the job completes with a FAILURE or TERMINATED status, the box terminates. Define additional conditions for the other jobs in the box in case the box is terminated.

job_terminator: y

Specifies that if the job's containing box completes with a FAILURE or TERMINATED status, the job terminates. You must add this attribute to each job definition that you want to terminate upon box failure.

More information:

Job Flow with Box Terminator Attribute (see page 188)
Job Flow with Job Terminator Attribute (see page 187)

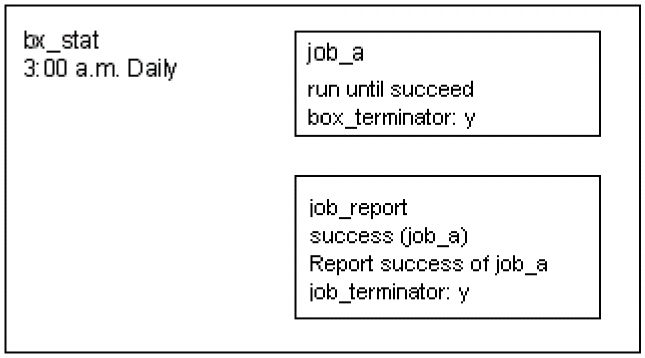
Time Conditions in a Box

Each job in a box runs only once each time the box runs. Therefore, do not define more than one time attribute for any job in a box because the job only runs the first time. If you want to put a job in a box, but you also want it to run more than once, you must define multiple start time conditions for the box itself, and define no time conditions for the job.

Note: The box must be running before the job can start. Do not assign a start time for a job in a box if the box will not be running at that time. If you do, the next time the box starts, the job starts immediately.

Example: Define Time Conditions for a Box Job

The following illustration shows a scenario with time conditions in a box:



In the illustration, job_a is defined to run repeatedly until it succeeds; job_report has one starting condition, the success of job_a.

At 3:00 a.m., bx_stat starts running, which causes job_a to start running. If job_a is successful, job_report runs and also succeeds.

If job_a fails, it will not be able to run again until the next time the box starts because jobs run only once per box execution. In this situation, the following occurs:

- Job job_report is still ACTIVATED while it waits for the success of job_a, and the status of the box is RUNNING.
- Because job_a is defined as a box terminator, the box then enters into a TERMINATED state.
- This change also terminates job job_report because its job_terminator attribute is set to y.
- Box bx_stat is now in a state that permits it to run again at 3:00 a.m. the following day.

If job_a was not defined as a box terminator, the box remains in RUNNING state indefinitely.

Force Jobs in a Box to Start

You can use the sendevent command to send a FORCE_STARTJOB event to force a job to start, even if its starting conditions have not been met.

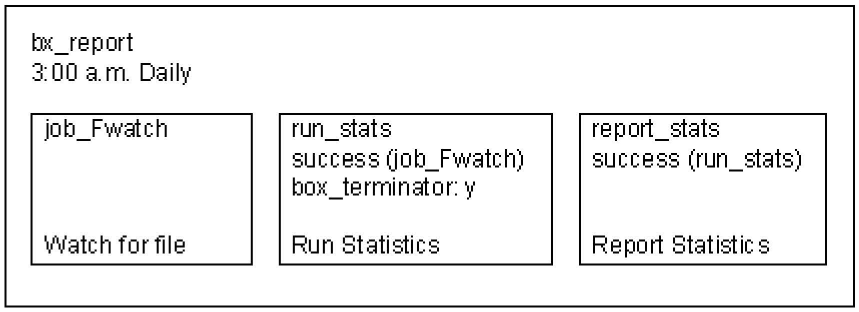
You can also execute the FORCE_STARTJOB command by selecting the Force Start Job button in the Job Activity Console, which is part of the CA WCC GUI.

Example: Force a Job in a Box to Start

This example defines a sendevent command that sends a FORCE_STARTJOB event to force a job in a box to run. You could use the following command to force the job run_stats to start:

```
sendevent -E FORCE_STARTJOB -J run_stats
```

In the following illustration, the box job bx_report contains three jobs (job_Fwatch, run_stats, and report_stats). If the job run_stats fails, the bx_report box job terminates because run_stats has a box_terminator attribute. If you force start run_stats, and it completes successfully, report_stats would still not start because the box it is in is not running.



Box Job Flow Examples

This section contains examples to help explain the flow of box jobs and the jobs they contain. These scenarios will help provide a clearer understanding of box job flow concepts.

Default Box Success and Box Failure

This scenario describes the default job flow for box job success and failure.

The box job do_statistics runs every day at 3:00 a.m. It contains three jobs:

update_accounts

Updates files. This job starts when do_statistics starts running. It has no other starting conditions.

run_stats

Runs statistics. This job starts when update_accounts completes successfully. It has no other starting conditions.

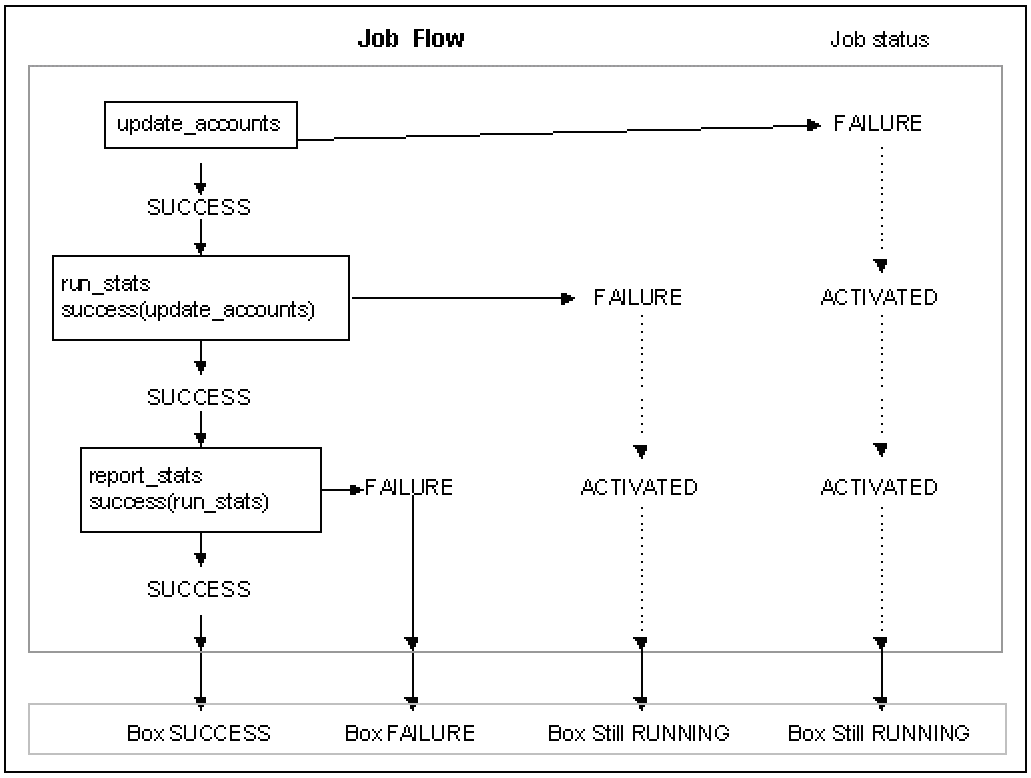
report_stats

Reports statistics. This job starts when run_stats completes successfully. It has no other starting conditions.

No conditions for success or failure have been defined for do_statistics; therefore the default conditions are applied. The box job completes successfully when all the jobs it contains have run and completed successfully. The box job fails when all jobs in the box have run and at least one has failed. The box job remains in the RUNNING state until all the jobs it contains have run.

The following illustration shows this job flow:

box_name “do_statistics"



Explicit Box Success and Box Failure

This scenario provides an example job flow in which specific conditions are defined for the success or failure of a box job.

The box job do_statistics runs every day at 3:00 a.m. It contains three jobs:

update_accounts

Updates files. This job starts when do_statistics starts running. It has no other starting conditions.

run_stats

Runs statistics. This job starts when update_accounts completes successfully. It has no other starting conditions.

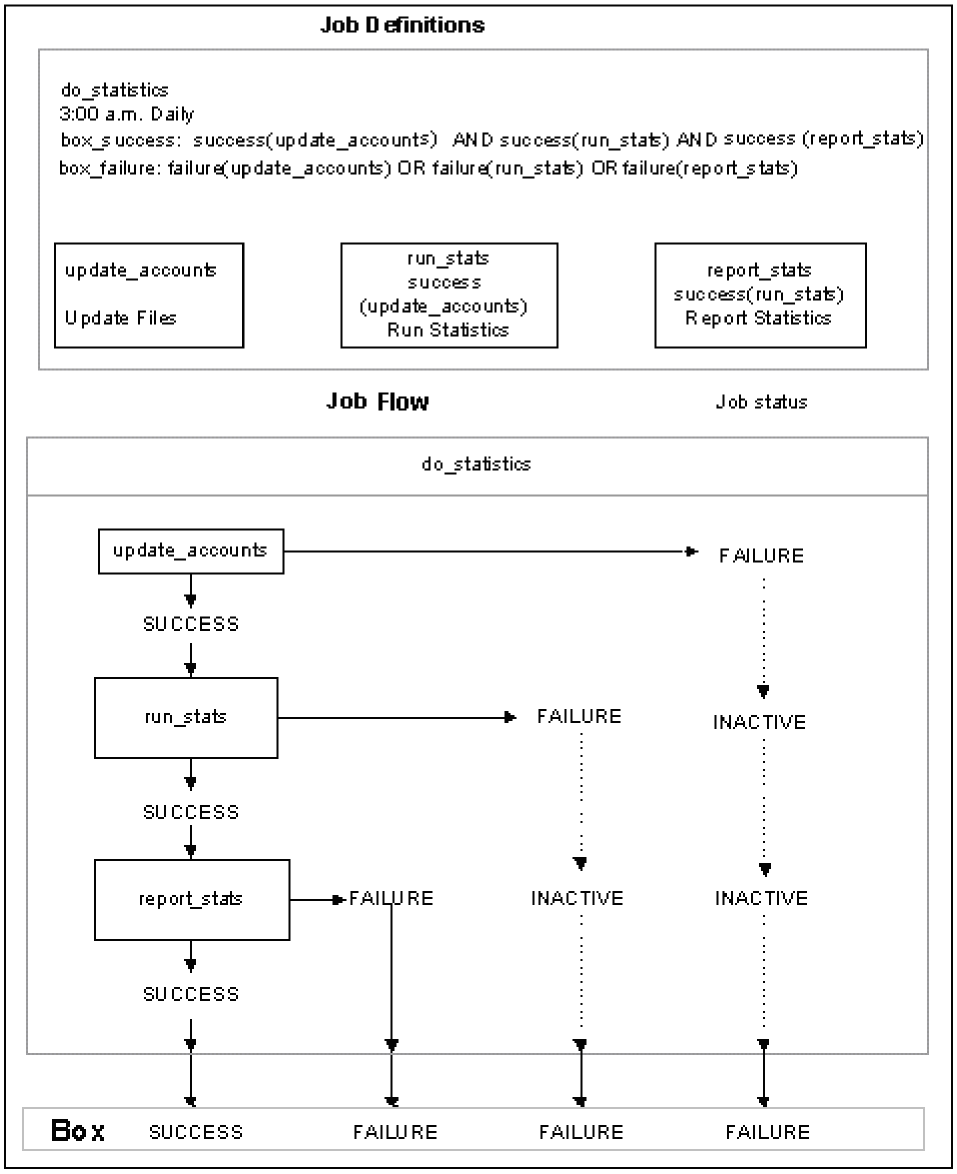
report_stats

Reports statistics. This job starts when run_stats completes successfully. It has no other starting conditions.

The following conditions define the criteria for success or failure of the box job do_statistics:

- The box job can complete successfully only when all of the jobs it contains have completed successfully.
- The box job fails if any of the jobs it contains fails.

The following illustration shows the job definitions and the job flow:



Job Flow with Job Terminator Attribute

This scenario provides an example job flow in which the job_terminator attribute is defined for a job in a box job.

The box job daily_accounts runs every day at 3:00 a.m. It contains two jobs:

daily_receipts

Processes receipts. This job runs when daily_accounts starts because it has no other starting conditions.

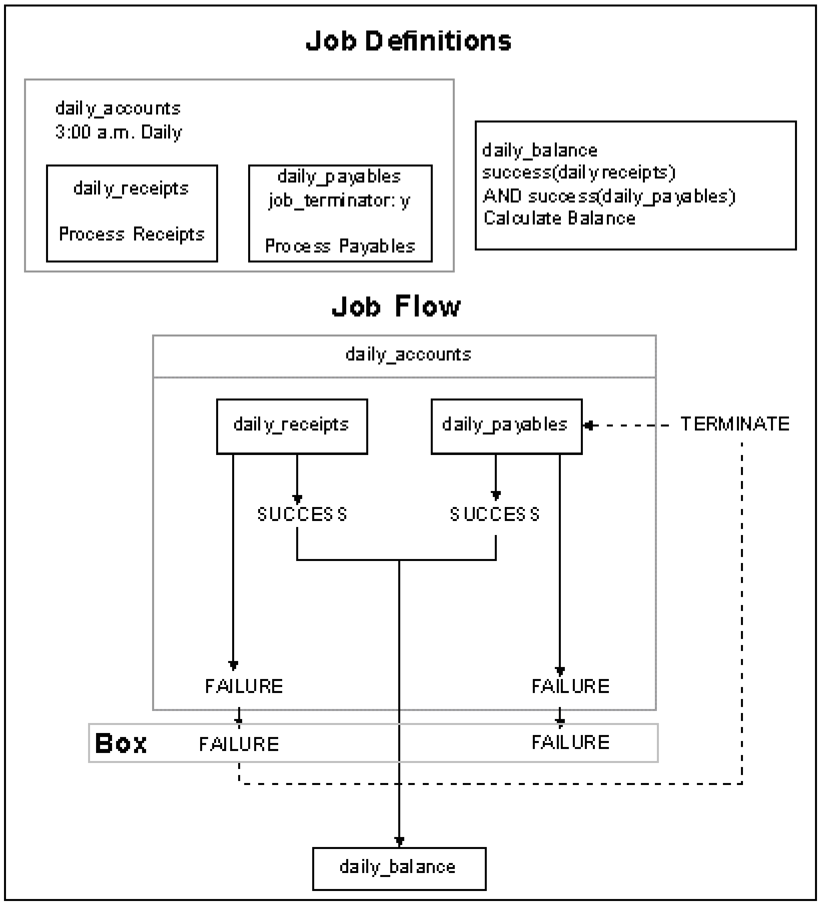
daily_payables

Processes payables. This job runs when daily_accounts starts because it has no other starting conditions. Because daily_payables includes a job_terminator attribute, daily_account is terminated if this job fails.

A third job, daily_balance, is not contained in daily_accounts and runs only if both daily_receipts and daily_payables complete successfully.

Because daily_accounts can only complete successfully if both of the jobs it contains complete successfully, the failure of daily_receipts causes daily_accounts to fail. This in turn triggers the job_terminator attribute in daily_payables, which terminates the job if the box that contains it fails.

The following illustration shows the job definitions and the job flow:



Job Flow with Box Terminator Attribute

This scenario provides an example job flow in which the box_terminator attribute is defined for jobs in a box job.

The box job daily_accounts runs every day at 3:00 a.m. It contains two jobs:

daily_receipts

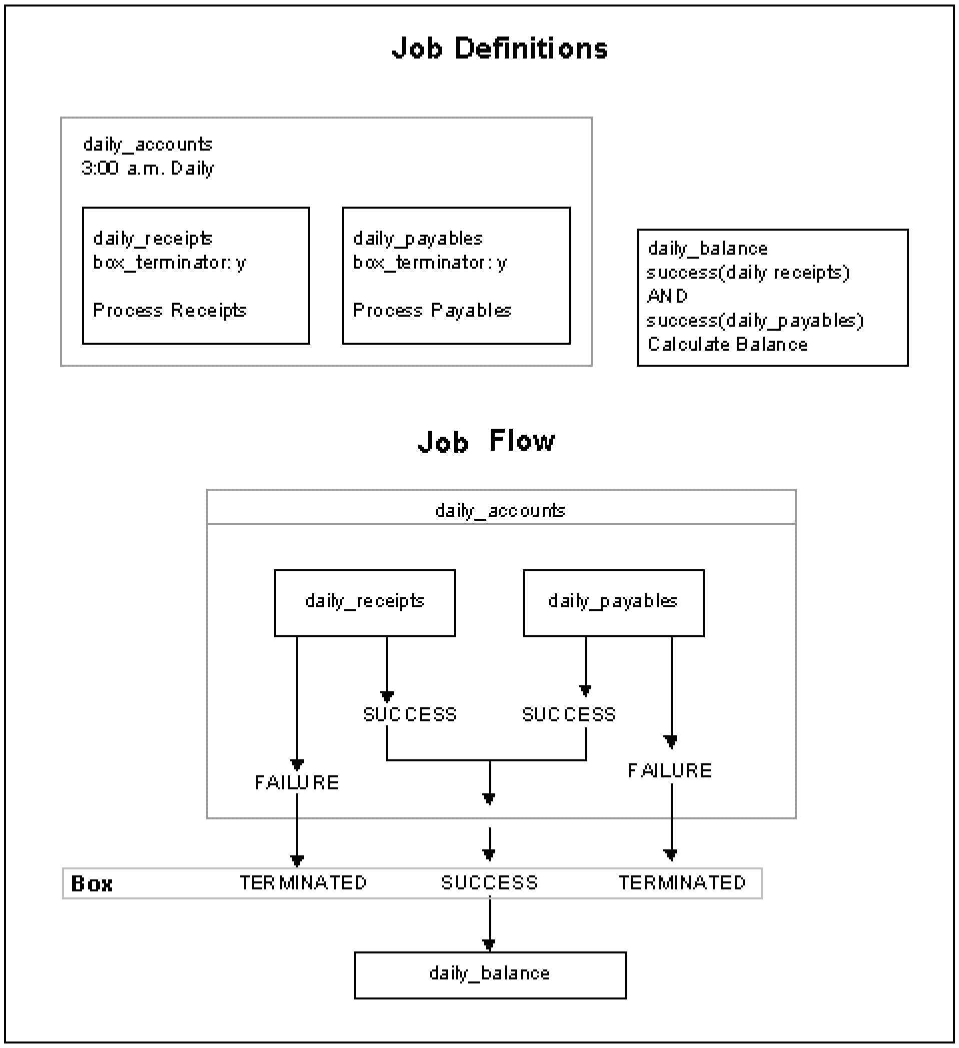
Processes receipts. This job runs when daily_accounts starts because it has no other starting conditions. Because daily_receipts includes a box_terminator attribute, daily_accounts will be terminated if this job fails.

daily_payables

Processes payables. This job runs when daily_accounts starts because it has no other starting conditions. Because daily_payables includes a box_terminator attribute, daily_accounts will be terminated if this job fails.

A third job, daily_balance, is not contained in daily_accounts and will run only if both daily_receipts and daily_payables complete successfully.

The following illustration shows the job definitions and the job flow:

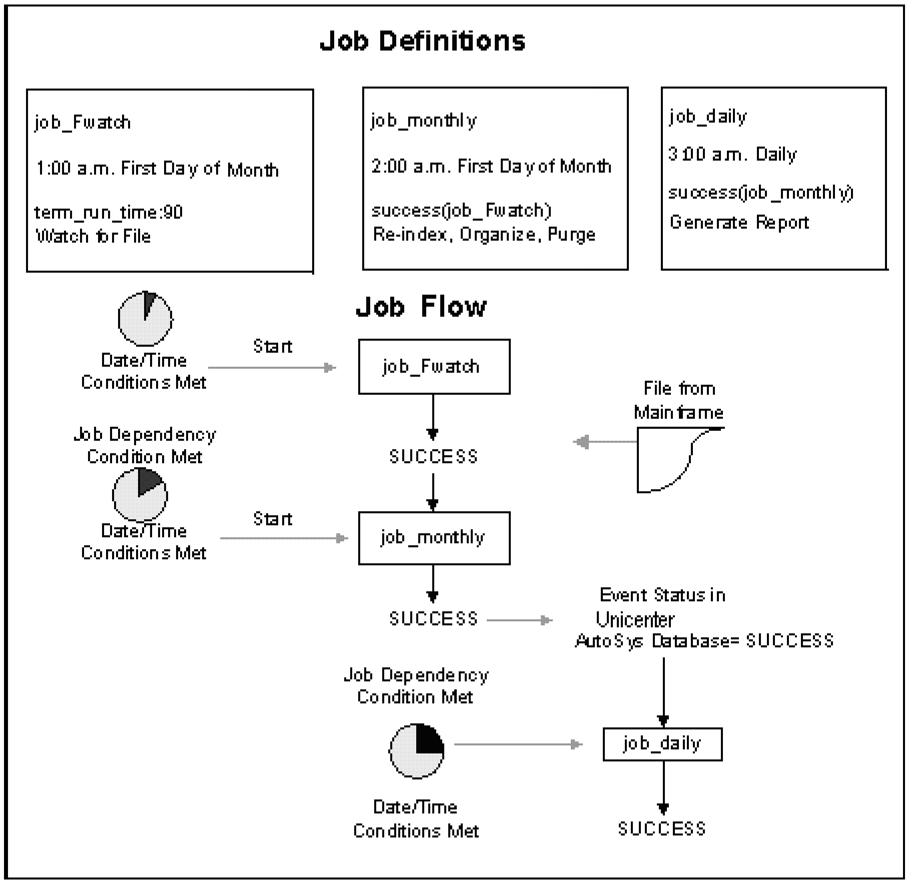


Advanced Job Flows

This section contains examples to help explain the flow of box jobs and the jobs they contain in advanced situations. These scenarios help provide a clearer understanding of advanced job flow concepts.

Job Flow with Time Conditions Running on the First of the Month

This scenario is an example of a job flow that begins on the first of every month.



The job flow consists of three jobs:

job_Fwatch

Waits for a specific file to be created by some mainframe process. This job runs at 1:00 a.m. on the first of every month and waits for 90 minutes before giving up.

job_monthly

Re-indexes, organizes, and purges its records based on the file created by the mainframe. This job runs at 2:00 a.m. on the first of the month only when job_Fwatch completes successfully.

job_daily

Generates a report. This job runs daily at 3:00 a.m. when job_monthly completes successfully.

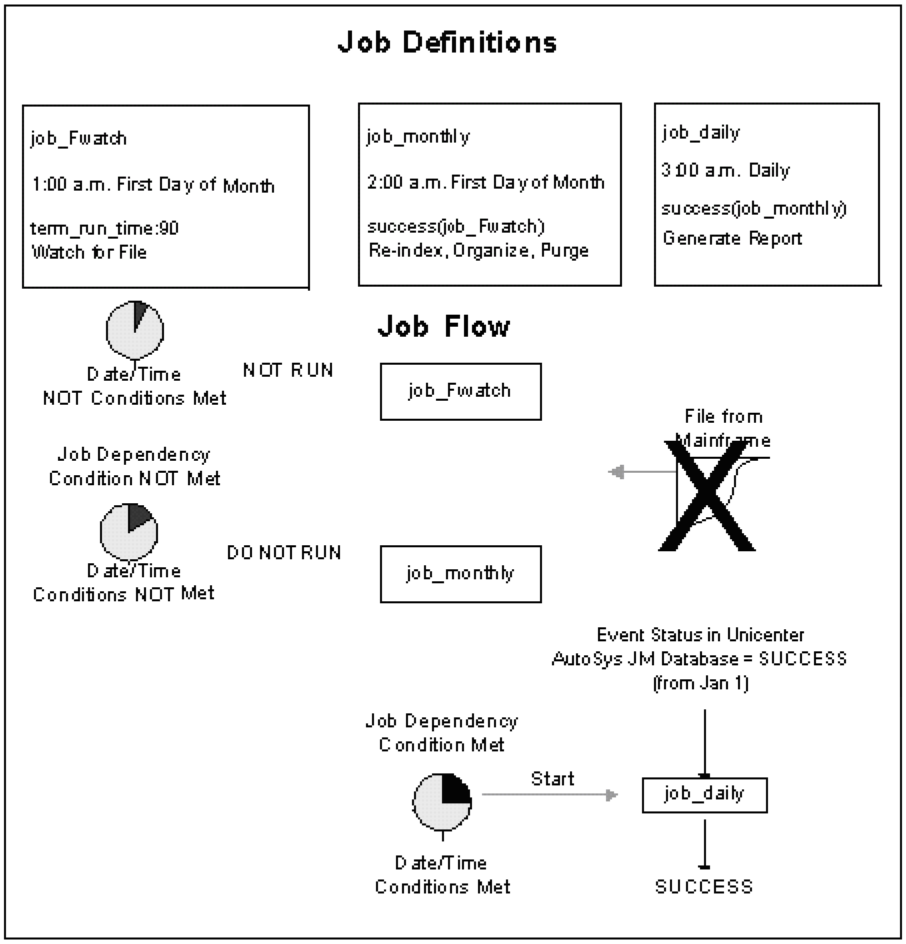
The failure of job_Fwatch causes job_monthly to skip its scheduled run because job_monthly can only complete successfully if job_Fwatch completes successfully. Job job_daily only runs if job_monthly completes successfully. By the same logic, job_daily always runs if job_monthly was able to successfully run at least once.

Note: The first time the cycle is run (for example, January 1), statuses behave as expected.

Job Flow with Time Conditions Running on the Second of the Month

This scenario builds upon the previous scenario and takes place on the following day.

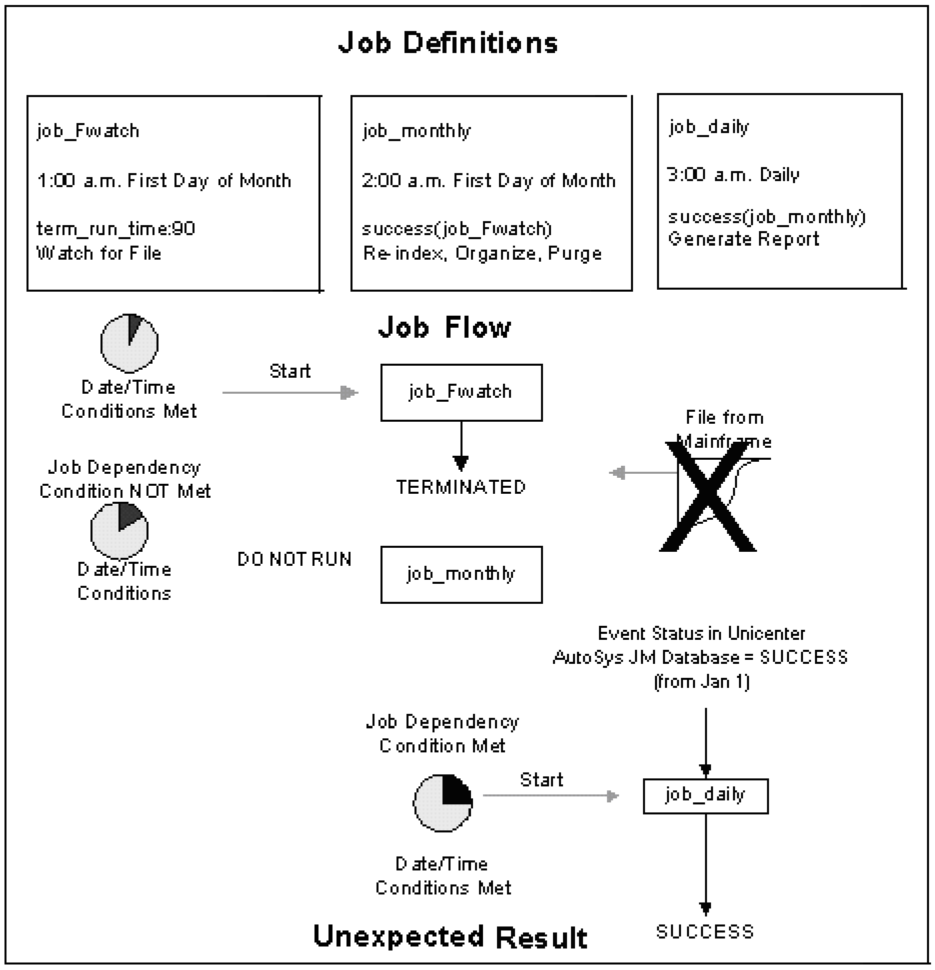
On days of the month other than the first, job_Fwatch and job_monthly do not run. They still have a status of SUCCESS in the database from the previous run on the first day of the month. As a result, job_daily still runs.



Job Flow with Time Conditions Running on the First of the Following Month

This scenario builds upon the previous scenario and takes place on the first day of the following month.

On the first day of the next month (for example, February 1), the file from the mainframe fails to arrive in the 90 minute wait time; therefore, job_Fwatch self-terminates. As a result, job_monthly misses its run for the month. However, because its event status in the database is still SUCCESS from the previous month, job_daily is able to run every day this month. When job_daily runs, it uses the previous month's data leading to invalid reports for the month.

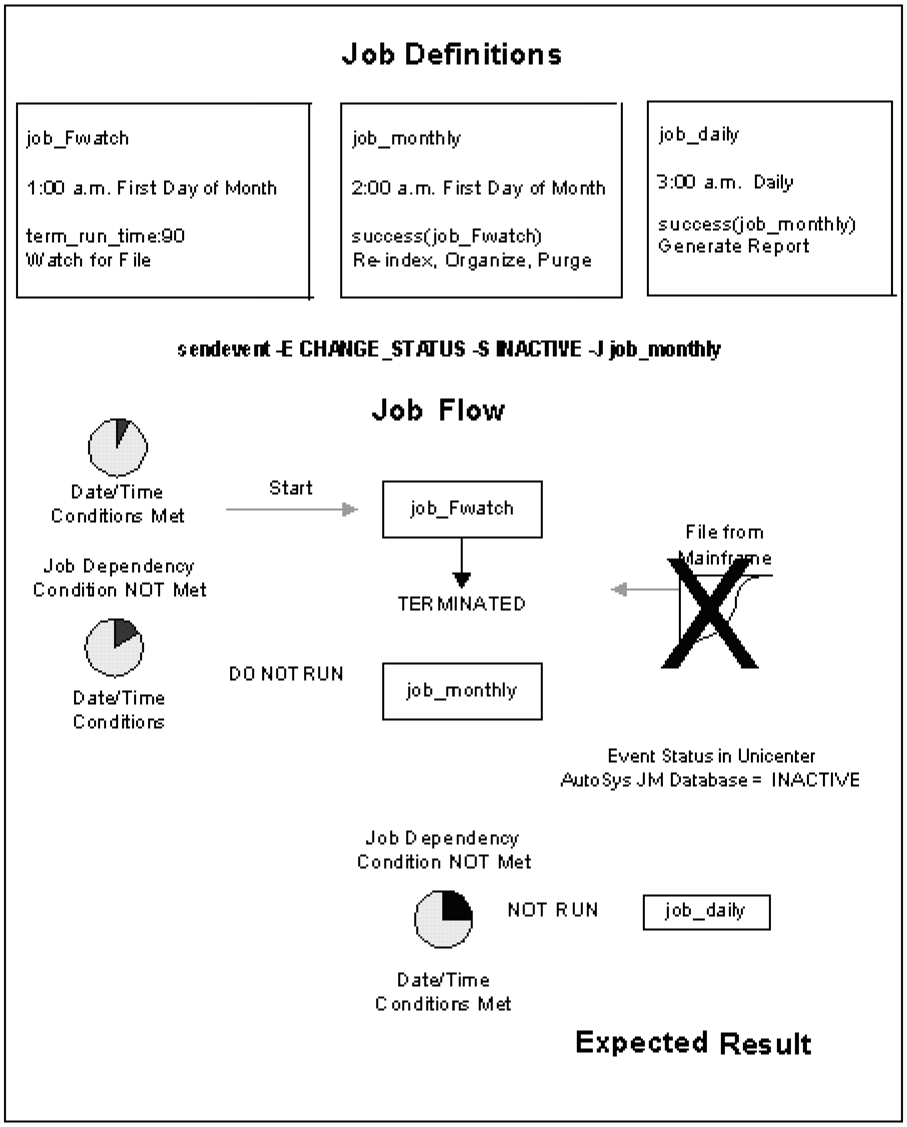


Resetting a Job Flow with Time Conditions Through INACTIVE Status Change

This scenario builds upon the previous scenario and takes place on the last day of the month.

To fix time-related statuses, you can use a sendevent command to change them to INACTIVE at the end of their valid interval. You can create another job to do this automatically.

Changing the status of job_monthly to INACTIVE at the end of every month allows job_daily to run only in the months that job_monthly completes successfully. In the following example, when job_Fwatch fails, job_monthly will not run, job_daily will not run because its status has been reset to INACTIVE.



Resetting a Job Flow with Time Conditions Through Box Job

This scenario builds upon the previous scenarios and takes place on the first day of the month.

Instead of issuing a sendevent command to change the status of the jobs, you can put the monthly process in a box, and set the box_failure or box_terminator attribute appropriately.

The job flow now consists of a box called box_monthly that runs at 1:00 a.m. on the first day of every month with the following jobs:

job_Fwatch

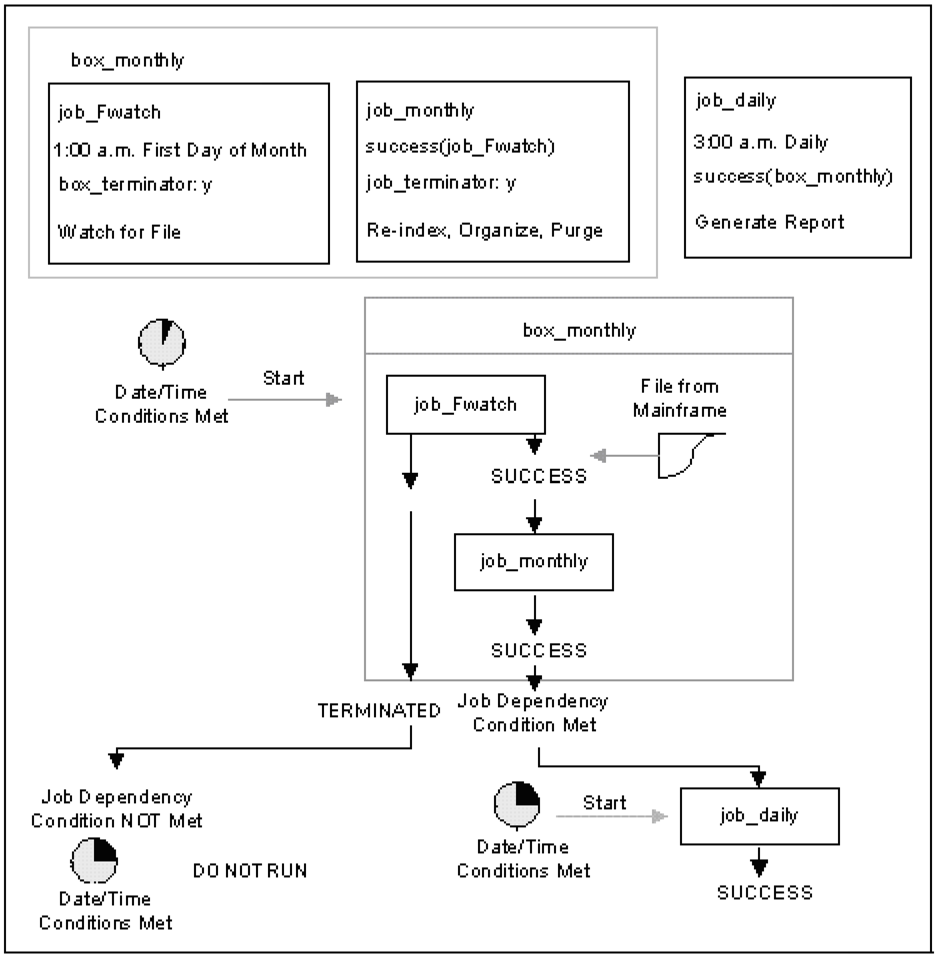
Waits for a specific file to be created by some mainframe process. This job runs at 1:00am on the first of every month and waits for 90 minutes before giving up.

job_monthly

Re-indexes, organizes, and purges its records based on the file created by the mainframe. This job runs at 2:00 a.m. on the first of the month only when job_Fwatch completes successfully.

A third job, job_daily, is not contained in box_monthly and runs only if job_Fwatch and job_monthly complete successfully.

The failure of job_Fwatch causes box_monthly to terminate because box_monthly can only complete successfully if both of the jobs it contains complete successfully. This in turn triggers the job_terminator attribute in job_monthly, which terminates the job if the box that contains it fails.



How a Box Job Is Created

Box jobs are a convenient way to start multiple jobs. When you put jobs in a box, you only have to start a single job (the box) for all the jobs in the box to start running.

Assume you want to schedule a group of jobs to start running when a file watcher job completes successfully. Instead of making each job dependent on the file watcher job, you can create a box job that is dependent on the file watcher job, remove the file watcher job dependency from the individual jobs, and put all of those jobs in the box. When the file watcher job completes successfully, the box job starts, which in turn starts all of the jobs it contains.

Note: For more information, see the *Reference Guide*.

Example: Creating a Box Job

This example shows how to define a box job named EOD_box that depends on the success of a file watcher job to run:

```
insert_job: EOD_box
job_type: box
condition: success(EOD_watch)
```

This JIL script instructs CA Workload Automation AE to do the following:

- Add a new job named EOD_box.
- Define the job as a box job.
- Run the job only if the file watcher job named EOD_watch completes with a SUCCESS status.

Box Job Attributes

The following attributes are required for all box job definitions:

box_name

Defines the name used to identify the job to CA Workload Automation AE. This name is used by other jobs as the name of their parent box.

job_type: BOX

Specifies that the job type is box.

Note: If you omit this attribute, the job type defaults to cmd (Command job).

condition

Defines the dependency conditions that must be met for the job to run.

Note: The condition attribute is not alwaya required, for example when a job is always started manually.

How Job Groupings Are Created

Box jobs provide one method of grouping jobs, but are typically used when all the jobs in the box share the same starting condition. CA Workload Automation AE provides the group and application attributes so you can logically group sets of jobs and boxes with unrelated starting conditions or dependencies. By specifying both group and application attributes in a job definition, you can make the job belong to both a group logical set and an application logical set.

Note: For more information, see the *Reference Guide*.

Example: Associate Jobs with Groups and Applications

This example shows how you can associate jobs with specific groups and applications to control processing.

Assume you want to create a set of jobs that run a suite of applications called EmployeePay that is used to manage employee salaries. The Accounting and Human Resources groups each have their own jobs defined to use the EmployeePay applications. The following JIL script defines two jobs (HR_payroll and ACCT_salaryreport):

```
insert_job: HR_payroll
job_type: cmd
...
group: HumanResources
application: EmployeePay
insert_job: ACCT_salaryreport
job_type: cmd
...
group: Accounting
application: EmployeePay
```

This JIL script instructs CA Workload Automation AE to do the following:

- Add two new command jobs (HR_payroll and ACCT_salaryreport).
- Associate job HR_payroll with the HumanResources group and the EmployeePay application.
- Associate job ACCT_salaryreport with the Accounting group and the EmployeePay application.

To run a job associated with the EmployeePay application, enter the following:

```
sendevent -e STARTJOB -I EmployeePay
```

To run a job associated with the Accounting group, enter the following:

```
sendevent -e STARTJOB -B Accounting
```

To run a job associated with both the EmployeePay application and Accounting group (intersection of both sets), enter the following:

```
sendevent -e STARTJOB -I EmployeePay -B Accounting
```

How an Existing Job Is Put in a Box

To place an existing job in a box, verify that the job is not running, and do either of the following:

- Use the update_job subcommand to change the current job definition.
- Use the delete_job subcommand to delete the current job definition, and use the insert_job subcommand to redefine the job.

  This method is useful when the job definition contains many non-default attributes that you want to deactivate instead of resetting them. However, if you delete and redefine the job, you must redefine any non-default attributes you want to keep from the previous definition.

Note: For more information, see the *Reference Guide*.

Example: Put an Existing Job in a Box

This example shows how to update the definition of an existing job to include it in a box.

The following JIL script uses the update_job subcommand to change the EOD_post job to put it in the EOD_box job:

```
update_job: EOD_post
box_name: EOD_box
```

This JIL script instructs CA Workload Automation AE to do the following:

- Update the job named EOD_post.
- Put the job named EOD_post in the box named EOD_box.

Delete a Box Job

To delete a box and every job it contains, enter the delete_box subcommand followed by the name of the box job to delete. For example, to delete the box EOD_box and every job in it, you would enter the following:

```
delete_box: EOD_box
```

To delete a box without deleting the jobs it contains, enter the delete_job command followed by the name of the box job to delete. The jobs in the box become stand-alone jobs. For example, to delete the box EOD_box without deleting the jobs in it, you would enter the following:

```
delete_job: EOD_box
```

Chapter 7: Command Jobs

This section contains the following topics:

Command Jobs

Command jobs let you run workload on UNIX and Windows client computers. On UNIX, you can define jobs to run scripts and commands. On Windows, you can define jobs to run command files and batch files.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows.

When you define a Command job, you can specify settings including the following:

Starting Conditions

Defines conditions (for example, date, time, job state, and machine state) that must be met before a job can start.

Disk Space Criteria

Defines the amount of free space required before a process can start. If the free space is not available, an alarm is sent and the job is rescheduled.

Job Profile

Specifies a script to be sourced that defines the environment where the command runs.

Note: On Windows, you can define a job profile using the Job Profiles - CA Workload Automation AE Administrator window in the Administrator utility.

Environment Variables

Specifies variables that define the local environment where the job runs.

User-defined Exit Codes

Defines exit codes to indicate job success and job failure. By default, an exit code of 0 (zero) indicates job success and any other code indicates job failure. When the job completes, the exit event (either SUCCESS or FAILURE) and program's exit code are stored in the database.

Standard I/O Files

Specifies the standard input, output, and error files.

The Directory the Job Runs Under (UNIX)

The directory that a Command job runs under on UNIX is determined by the following settings:

oscomponent.initialworkingdirectory Agent Parameter

Specifies the default initial working directory for all scripts. Options are the following:

- SCRIPT—Sets the path to where the script resides.
- USER—Sets the path to the home directory of the owner of the script.
- PATH—Sets the path to an absolute path to where the script runs.

If you do not specify a value, the parameter defaults to the path where the running cybAgent resides.

You can specify the oscomponent.initialworkingdirectory parameter in the agent's agentparm.txt file.

HOME and PWD Environment Variables

Overrides the oscomponent.initialworkingdirectory default directory. The directory depends on the following settings:

- If you specify a value for PWD (Present Working Directory) in the job definition, the job runs under the PWD you specified.
- If you specify a value for HOME in the job definition, but you do not specify a value for PWD, the job runs under the HOME directory.
- If you specify values for HOME and PWD in the job definition, the job runs under the PWD directory.

Determining Which Shell is Used (UNIX)

A shell is a program that provides an environment for users to interact with the system. Shell programs are files that contain a series of user commands that are grouped, using logic provided by the shell, to accomplish a task.

Shells can be defined by you, the script's programmer, your agent administrator, and your UNIX administrator. When you define a job that runs on UNIX, you may want to know which shell is used to run the script because different shells have different facilities and characteristics. Some functions are specific to certain shells and may be incompatible with other shells.

The shell the CMD job uses is determined by the following settings in the following order:

1. The shell attribute (if specified in the job definition)
2. The oscomponent.defaultshell parameter in the agentparm.txt file (if not specified in the shell attribute)
3. The user default shell defined in the user profile (if one exists on the computer)
4. On UNIX, the shell sourced from the /etc/auto.profile script (if one is specified)
5. The shell directive line (first line) of the script (if one is specified)

Notes:

- If the shell is defined in the first line of the script, you do not need to include the shell attribute in the job definition. The shell attribute tells the agent which shell interpreter to use when launching the command. When the command executes a script that specifies a shell, the script runs from that point forward under the shell specified in the script.
- If the oscomponent.checkvalidshell parameter in the agent's agentparm.txt file is set to true (the default), all shells that are used must be specified using the oscomponent.validshell parameter. The path defined in the first line of the script or in the job definition must match the corresponding path in the oscomponent.validshell parameter. If the shell you want to use is not defined on the agent, the job fails. For more information about specifying valid shells, see the oscomponent.checkvalidshell and oscomponent.validshell parameters in the *CA Workload Automation Agent for UNIX, Linux, or Windows Implementation Guide*.

Shell Initialization Files (UNIX)

When you log in to the UNIX operating system, the operating system reads a series of initialization files depending on the shell you use.

C Shell Initialization Files

When you log in to the C shell, it automatically runs a number of files. The first file run is a system file named /etc/.login, which contains system-wide configuration information (such as your default path). After these files are run, the C shell reads and runs the commands from the following files in your home directory:

.cshrc

Establishes variables and operands that are local to a specific shell. Each time you create a new shell, the C shell reinitializes these variables for the new shell.

The following is a sample .cshrc file:

```
set noclobber
set ignoreeof
set path = (~/bin $path /usr/games)
alias h history
```

.login

Contains commands that you want to run once at the beginning of each session. If the C shell is running as a login shell, it reads and runs the commands in the .login file in your home directory after login.

The following is a sample .login file:

```
setenv history 20
setenv MAIL /usr/spool/mail/$user
```

.logout

Runs when you exit from your login shell.

Korn Shell Initialization Files

The Korn shell supports three startup scripts:

/etc/profile

Contains system-wide startup commands. This file is a login script and runs when you log in.

$HOME/.profile

Runs when you log in. Use this login script to set options, set the path, and set and export variable values.

The following is a sample $HOME/.profile file:

```
set -o allexport
PATH=.:/bin:/usr/bin:$HOME/bin
CDPATH=.:$HOME:$HOME/games
PS1='! $PWD> '
ENV=$HOME/.kshrc
TIMEOUT=0
set +o allexport
```

The script named in the Korn shell variable ENV

Runs when you create a Korn shell or run a Korn shell script. Use this file to define aliases and functions and to set default options and variables that you want to apply to the Korn shell invocation.

Bourne shell initialization files

When you log in to your system, the Bourne shell looks for the /etc/profile file. This file contains system-wide startup commands for the Bourne shell. Only the system administrator can change this file.

After running the commands in the /etc/profile file, the Bourne shell runs the commands in the $HOME/.profile file. Therefore, you can override the system-wide commands in the /etc/profile file with commands in the $HOME/.profile file.

Define a Command Job

You can define a Command (CMD) job to run workload on UNIX and Windows client computers. The job can run a script, execute a UNIX command, or run a Windows command file.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows.

To define a Command job

1. Insert a job and specify the following attributes in the definition:

   job_type: CMD

   Specifies that the job type is Command.

   Note: For backwards compatibility with previous versions of CA Workload Automation AE, you can specify **job_type: c**.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   command

   Specifies the command, executable, UNIX shell script, application, or batch file to run when all the starting conditions are met.

2. (Optional) Specify optional Command attributes:

   - blob_file
   - blob_input
   - chk_files
   - envvars
   - fail_codes
   - heartbeat_interval
   - interactive
   - max_exit_success
   - shell
   - std_err_file
   - std_in_file
   - std_out_file
   - success_codes
   - ulimit

3. (Optional) Specify common attributes that apply to all job types.

   The Command job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Run a Command on UNIX

This example runs the /bin/touch command on the file named /tmp/test_run.out. The job runs on the UNIX client computer named unixagent.

```
insert_job: test_run
job_type: CMD /* This attribute is optional for Command jobs. CMD is the default. */
machine: unixagent
command: /bin/touch /tmp/test_run.out
```

Example: Run a Command File on Windows

This example runs the c:\bin\test.bat command on the Windows client computer named winagent. The command is enclosed in quotation marks because the path contains a colon.

```
insert_job: test_run
job_type: CMD /* This attribute is optional for Command jobs. CMD is the default. */
machine: winagent
command: "c:\bin\test.bat"
```

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following Command job attributes have default values:

fail_codes

Defines which exit codes indicate job failure.

Default: Any exit code other than 0 (zero) indicates job failure

interactive

(Windows only) Specifies whether to run a Command job in interactive mode or in batch mode on Windows.

Default: n (The job runs in batch mode.)

max_exit_success

Defines the maximum exit code that the job can exit with and be considered a success.

Default: 0 (The job interprets only zero as job success.)

owner

Specifies the user ID that the job runs under.

Default: The default owner (the user ID who invokes jil to define the job)

shell

(UNIX only) Specifies the name of the shell used to execute the script or command file.

Note: Alternatively, you can override the default shell by specifying the shell in the first line of the script. If a shell is specified in the script and in the job definition, the job uses the shell specified in the job definition.

Default:

- oscomponent.defaultshell agent parameter, if specified
- The user default shell defined in the user profile, if the shell is not specified in the job definition, script, or oscomponent.defaultshell parameter

std_err_file

Defines the path and file name where you want to redirect all standard error output.

CA WA Agent Default: The agent's spool directory (*installation_directory*/SystemAgent/*agent_name*/spool)

Legacy Agent Default: /dev/null (UNIX) or NULL (Windows)

std_out_file

Defines the path and file name where you want to redirect all standard output.

CA WA Agent Default: The agent's spool directory (*installation_directory*/SystemAgent/*agent_name*/spool)

Legacy Agent Default: /dev/null (UNIX) or NULL (Windows)

success_codes

Defines which exit codes indicate job success.

Default: 0 (The job interprets zero as job success.)

Notes: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Override the Default Shell Using the shell Attribute

This example overrides the default shell using the shell attribute. The job uses the C shell run the test1.csh script.

```
insert_job: unix_job1
job_type: CMD
machine: unixagent
command: /mfg/test1.csh
shell: /bin/csh
```

Note: This job succeeds only if the user's default shell is csh. To set the default shell, define the following parameters in the agent's agentparm.txt file:

```
oscomponent.defaultshell=/bin/csh
oscomponent.defaultshell.force=true
```

Example: Override the Default Shell by Specifying the Shell in a Script

This example overrides the default shell with the C shell specified in the following test1.csh script:

```
#!/bin/csh -f
if ( $LOGNAME != guest) then
    echo "User is not guest"
endif
echo $LOGNAME logon
exit 0
```

The following job definition runs the test1.csh script:

```
insert_job: unix_job2
job_type: CMD
machine: unixagent
command: /mfg/test1.csh
owner: guest
```

The shell attribute is not required in the job definition because the shell is specified in the first line of the script.

Example: Override the Default Background Mode Using the interactive Attribute

This example runs a Command job in interactive mode on Windows. The job opens the config.txt file in the Windows notepad application on the Windows desktop.

```
insert_job: edit_file
job_type: CMD
machine: winagent
description: "Edit/review a configuration file"
command: notepad.exe "c:\run_info\config.txt"
interactive: y
```

More information:

Determining Which Shell is Used (UNIX) (see page 204)

Verify File Space Before a Job Starts

You can define a Command job to check if one or more UNIX file systems or Windows drives have the required amount of available space. At run time, the agent checks whether the required space is available on the machine where the job runs. If the requirements are met, the job starts. If the requirements are not met, the agent generates an alarm and the job does not start. The system tries to verify the file space again and start the job. The number of tries is determined using the n_retrys attribute. If the n_retrys attribute is not specified in the job definition, the number of tries is determined by the MaxRestartTrys parameter in the configuration file (UNIX) or the Max Restart Trys field in the Administrator utility (Windows). If the required space is still not available after all the restart attempts, the job fails.

By default, jobs do not check the available file space.

To verify file space before a job starts

1. Define a Command job (see page 207).
2. Add the following attribute to the job definition:

   chk_files

   Specifies the required amount of space on one or more file systems (UNIX) or drives (Windows) in kilobytes (KB).

3. Run the job.

   The file space is verified before the job starts.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Verify the Available File Space on UNIX

This example checks whether the file system named roots has 100 KB of available space. This example also checks whether the file system named auxfs1 has 120 KB of available space. The specified file space must be available before the job can start.

```
insert_job: unix_chk
job_type: CMD
machine: unixagent
command: /u1/procrun.sh
chk_files: /roots 100 /auxfs1 120
```

Example: Verify the Available File Space on Windows

This example checks whether the C: drive has 100 KB of available space and the D: drive has 120 KB of available space. The specified file space must be available before the job can start.

```
insert_job: win_chk
job_type: CMD
machine: winagent
command: "C:\Programs\Payroll\pay.exe"
chk_files: "C: 100 D: 120"
```

Pass Positional Arguments in a Command Job

When running workload, you might need to pass data between jobs and across platforms. You can pass positional arguments to a command or script in your job definition. Positional arguments are variables that can be passed to a program at the time the program is invoked. The arguments are assigned in the order they are passed.

To pass positional arguments in a Command job

1. Define a Command job (see page 207).
2. Add the following attribute to the job definition:

   ```
   command: file argument...
   ```

   file

   Specifies the command, executable, UNIX shell script, application, or batch file to run when all the starting conditions are met.

   argument...

   Specifies one or more arguments to pass to the command or script at run time.

   Note: Separate each argument with a space. You must specify each argument in the order it is expected in the program.

3. Run the job.

   The positional arguments are passed to the program.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Pass Positional Arguments to a UNIX Script

This example passes three arguments to a UNIX script. The first argument passed is "user 1". This argument is enclosed with quotation marks because it contains a space. The second argument passed is 905-555-1212, and the third argument is 749.

```
insert_job: cmd_job1
job_type: CMD
machine: unixprod
command: addinfo.sh "user 1" 905-555-1212 749
```

Example: Pass Positional Arguments to a Windows Program

This example passes two data files to a Windows program. The arguments are enclosed with quotation marks because they contain spaces.

```
insert_job: cmd_job2
job_type: CMD
machine: winprod
command: "c:\Programs\Payroll\pay.exe" "C:\Pay Data\salary.dat" "C:\Pay
Data\benefits.dat"
```

Pass Environment Variables in a Command Job

You can specify environment variables to define the local environment the program runs in. You can modify existing environment variables or create your own.

To pass environment variables in a Command job

1. Define a Command job (see page 207).
2. Add the following attribute to the job definition:

   envvars

   Specifies the environment variables that define the local environment.
3. Run the job.

   The environment variables are passed to the program.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Pass Environment Variables to a UNIX Script

This example includes two envvars attributes that pass environment variables to a script and a third envvars attribute that defines the Present Working Directory (PWD). The parameter "user 1" is enclosed with quotation marks because it contains a space.

```
insert_job: unix_job
job_type: CMD
machine: unixprod
command: /home/scripts/pay
envvars: NAME="user 1"
envvars: JOB=PAYROLL
envvars: PWD=/usr/scripts/dailyrun
```

In this example, the pay script can reference these variables:

Environment Variable	Value Passed
NAME	user 1
JOB	PAYROLL
PWD	/usr/scripts/dailyrun

Note: If the parameter oscomponent.loginshell is set to true in the agent's agentparm.txt file, the agent invokes the user environment while running the script. To override the value of a shell variable that is already defined in the user login file, reassign a value to this variable in the script.

Example: Pass Two Environment Variables to a Windows Command File

This example runs the processrun.exe command file on a Windows computer named winprod. The job uses two local environment variables named PATH and TEMP.

```
insert_job: cmd_job1
job_type: CMD
machine: winprod
command: "c:\cmds\processrun.exe"
envvars: PATH="c:\windows\system32"
envvars: TEMP="c:\temp"
```

UNIX Environment Variables

When a Command job runs under a specific user account on UNIX, the agent can pass the user's environment variables to the script or program. You can also set up a script's running environment by overriding the environment variables in the job definition. For example, you can override the HOME environment variable to run the script under a user’s login directory.

You can pass the following UNIX environment variables in a job definition to override the variable values:

HOME

Identifies the user's login directory. You can override the HOME value to set up a user-specific environment by specifying a different login directory in the job definition.

Example: HOME=/home/guest/bin

Notes:

- You can set up the script's running environment in the .profile and .login files.
- You must also set the oscomponent.loginshell parameter to true in the agent's agentparm.txt file to run the login scripts located in the HOME directory.

PATH

Provides a list of directories that the shell searches when it needs to find a command. Each directory is separated by a colon. and the shell searches the directories in the order listed. The PATH variable is the most important environment variable. You can override the PATH value to set up a user-specific environment by specifying a different PATH in the job definition.

Note: Overriding the default system path can result in the "command not found" error.

ENV

Contains the name of the initialization file to run when a new Korn shell starts. You can override the ENV value to set up a user-specific environment by specifying a different ENV value in the job definition.

Example: ENV=/home/guest/bin/myenv

Note: The name of the file used to set up the script-running environment must be .profile. The .profile must be the same file used with the HOME variable.

PWD

Contains the absolute path name of your current directory.

Define Alternative Error, Input, and Output Sources

By default, the standard output and standard error output are redirected to the agent's spool directory. You can define alternative error, input, and output sources in your job definition that override these defaults.

To define alternative error, input, and output sources

1. Define a Command job (see page 207).
2. Add one or more of the following attributes to the job definition:

   std_err_file

   Specifies where you want to redirect the standard error output. The output can be redirected to a file or a blob.

   std_in_file

   Specifies where you want to redirect standard input from. The input can be a file or a blob.

   std_out_file

   Specifies where you want to redirect the standard output. The output can be redirected to a file or a blob.

3. Run the job.

   The alternative sources are defined.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Specify the Standard Input File and Output on UNIX

This example specifies the /tmp/test.in file as the standard input file and /tmp/test.out as the standard output file .

```
insert_job: unix_stdin
job_type: CMD
machine: unixagent
command: /usr/common/backup
std_in_file: /tmp/test.in
std_out_file: /tmp/test.out
```

Send a User-Defined Exit Code

By default, the scheduling manager interprets an exit code of 0 (zero) as job success and any other number as job failure. However, you can map exit codes other than 0 as job success and you can map specific codes as job failure.

To send a user-defined exit code

1. Define a Command job (see page 207).
2. Add one or more of the following attributes to the job definition:

   fail_codes

   Defines which exit codes indicate job failure.

   success_codes

   Defines which exit codes indicate job success.
3. Run the job.

   The user-defined exit code is sent.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Define a Range of Exit Codes to Indicate Job Failure

Suppose that you want a job named CMDJOB to run the procjob.exe file. The job is considered to have failed if the exit code is in the 40-50 range.

```
insert_job: CMDJOB
job_type: CMD
machine: winagt
command: "c:\temp\procjob.exe"
fail_codes: 40-50
```

Example: Define a Exit Code to Indicate Job Success

This example shows the first and last lines of the payroll.sh script. The script returns the self-defined exit code 100 to the scheduling manager.

```
#!/usr/bin/sh
.
.
.
exit 100
```

The following job definition runs the script. The success_codes attribute defines exit code 100 as success, indicating successful completion of the script.

```
insert_job: test_blob
job_type: CMD
machine: unixagent
command: /home/esp/payroll.sh
success_codes: 100
```

Create a Job Blob

You can create a job blob that is associated with the Command job you are defining. The blob can contain the data that you specify in the job definition (input job blob), the job's output (output job blob), or the job's error messages (error job blob). Input job blobs are uploaded to the database when the job is defined. Output and error job blobs are uploaded to the database after the job runs.

Other jobs on different computers can use the input job blobs and output job blobs. Jobs cannot use error job blobs as input.

To create a job blob

1. Define a Command job (see page 207).
2. Add one or more of the following attributes to the job definition:

   blob_file

   Specifies the file containing the data to insert for the blob.

   blob_input

   Specifies the text to insert for the blob.

   std_err_file

   Specifies that the job blob contains the job's standard error messages in textual or binary data format.

std_out_file

Specifies that the job blob contains the job's standard output messages in textual or binary data format.

3. Run the job.

   The job blob is created.

Notes:

- For more information about attributes and their JIL syntax, see the *Reference Guide*.
- You can also use the insert_blob subcommand to create an input job blob after a job is defined.

Example: Create an Input Job Blob at Job Definition Time

This example defines a CMD job that creates and uses an input job blob. When the job is defined, a blob is created using the text that is specified in the blob_input attribute. When the job runs, it uses the created blob as input and treats the blob data as textual data.

```
insert_job: test_blob
job_type: CMD
command: cat
machine: unixagent
blob_input: <auto_blobt>multi-lined text data for job blob
</auto_blobt>
std_in_file: $$blobt
owner: produser@unixagent
```

Example: Create an Input Job Blob Using a File at Job Definition Time

This example defines a CMD job that creates and uses an input job blob. When the job is defined, a blob is created using the textual data contained in the blob_input_file.txt file. The blob is associated with the job. When the job runs, it uses the created blob as input and treats the blob data as textual data.

```
insert_job: test_blob
job_type: CMD
machine: unixagent
command: cat
blob_file: /blob_input_file.txt
std_in_file: $$blobt
owner: produser@unixagent
```

Example: Create an Error Input Blob

This example captures the standard error output and stores it as a textual format blob associated with the job.

```
insert_job: report_job1
job_type: CMD
machine: localhost
command: myapplication
std_err_file: $$blobt
```

Example: Create an Output Blob

This example captures the standard output and stores it as a textual format blob associated with the job.

```
insert_job:  report_job1
job_type: CMD
machine: localhost
command: myapplication
std_out_file: $$blobt
```

Specify a Command or Script Name Without the Full Path

When defining a Command job, the agent usually requires the full path to the command or script name you want to run. However, you can specify the command or script name without the full path if the proper conditions are met.

To specify a command or script name without the full path

1. Define a Command job (see page 207).
2. Add the following attribute to the job definition:

   command

   Specifies the command, executable, UNIX shell script, application, or batch file to run when all the starting conditions are met.

   UNIX: To specify a name without the full path, the following conditions must be met:

   - The agent is running under the root account.
   - The agent is configured to resolve environment variables.
   - The user ID you enter in the owner attribute has the authority to run the job on the agent. The user default shell is used.
   - The path to the script or command name is set in the PATH system environment variable for the specified user ID.

Windows: To specify a name without the full path, the following conditions must be met:

- The agent is configured to search for paths to command or script files.
- The script or command file is located in *one* of the following directories:

  - The directory the agent is installed in

  - WINDOWS\system32 directory on the agent computer

  - WINDOWS\system directory on the agent computer

  - WINDOWS directory on the agent computer

  - Any other directory whose path is set in the system path or user path on the agent computer

3. Run the job.

   The command or script name is specified.

Notes:

- For more information about attributes and their JIL syntax, see the *Reference Guide*.
- To configure the agent to resolve environment variables, ask your agent administrator to refer to the information about the oscomponent.lookupcommand parameter in the *CA Workload Automation Agent for UNIX, Linux, or Windows Implementation Guide*.
- Environment variables are not currently supported in the command attribute for Command jobs that run on Windows.

Example: Run a Script that is Located in a Path Set in the PATH Variable (UNIX)

This example runs a script named procscript.sh. The job runs under the user ID jsmith, which has the authority to run the script. The path to procscript.sh is set in the PATH system environment variable for jsmith on the agent computer and the agent is configured to search for paths to command and script files.

```
insert_job: unix_job
job_type: CMD
machine: unixagent
command: procscript.sh
owner: jsmith
```

Example: Specify a Script Name Without the Full Path (Windows)

This example runs a script named procscript.bat. The path to procscript.bat is set in the system path on the agent computer and the agent is configured to search for paths to command and script files.

```
insert_job: win_job
job_type: CMD
machine: winagent
command: procscript.bat
```

Specify a Command or Script Name Using an Environment Variable (UNIX)

When defining a Command job on UNIX, you can specify the command or script name using an environment variable (for example, $MY_PATH/myscript.sh).

To specify a command or script name using an environment variable

1. Define a Command job (see page 207).
2. Add the following attribute to the job definition:

   command

   Specifies the command, executable, UNIX shell script, application, or batch file to run when all the starting conditions are met. Include the environment variable in the name.

   Note: To use an environment variable, the following conditions must be met:

   - The agent is running under the root account.
   - The agent is configured to resolve environment variables.
   - The user ID you enter in the owner attribute has the authority to run the job on the agent computer. The user default shell is used.
   - The environment variable you use, such as $MY_PATH, is set in the specified user ID's profile file.
3. Run the job.

   The command or script name is specified.

Notes:

- For more information about attributes and their JIL syntax, see the *Reference Guide*.
- To configure the agent to resolve environment variables, ask your agent administrator to refer to the information about the oscomponent.lookupcommand parameter in the *CA Workload Automation Agent for UNIX, Linux, or Windows Implementation Guide*.

Example: Run a Script that is Located in a Path Set in a User Environment Variable

In this example, an agent named unixagent runs a script named myscript.sh. The job runs under the user ID jsmith, which has the authority to run the script. The path to myscript.sh is set in the user environment variable $MY_PATH, which is defined in the profile file for jsmith. The agent is configured to search for paths to command and script files.

```
insert_job: unix_job
job_type: CMD
machine: unixagent
command: $MY_PATH/myscript.sh
owner: jsmith
```

Run a Script Under a Specific User Account (UNIX)

You can define a Command job to run a UNIX command or script under a specific user's account.

To run a script under a specific user account on UNIX

1. On the client computer where you want to run the job, do *one* of the following:
   - Start the agent as root.
   - Start the agent as a user other than root.

     Note: This user account must have permissions to access the resources that the job requires. If the job is defined to run under a different user (owner), the user account the agent is started under must have permissions to switch to that user. Otherwise, the job fails.
2. Define a Command job (see page 207).
3. Add the following attribute to the job definition:

   owner

   Specifies the user ID that the job runs under.

   Default: The default owner (the user ID who invokes jil to define the job)

   Note: The user ID must have the permissions to access the required directories and run the commands and scripts located on the agent computer.
4. Run the job.

   The script runs under the specified user.

Notes:

- If the shell is specified in the first line of the script and its path matches the path defined in the oscomponent.validshell parameter, you do not have to specify the shell attribute in the job definition to define the shell that you want the agent to use. For example, you can run a script using the environment defined in a specific user's account.
- For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Run a Script Under a Specific User Account

In this example, if the first line of the script is #!/bin/ksh and the path /bin/ksh is defined using the oscomponent.validshell parameter, you can use the owner attribute to run the script under a specific user account, as follows:

```
insert_job: unix_job
job_type: CMD
machine: unixagent
command: /home/guest/bin/cmd1.ksh
owner: guest
```

In this example, the agent runs the cmd1.ksh Korn script using the environment defined in either the $HOME/.login file or the
$HOME/.profile file, depending on which login shell is defined for user guest:

- If csh is defined, the $HOME/.login file is used.
- If ksh is defined, the $HOME/.profile file is used.

The login shell for user guest does not have to be the Korn shell. For example, the agent can pick up the following environment variables for user guest:

```
HOME=/home/guest
LOGNAME=guest
USER=guest
SHELL=/usr/bin/csh
PWD=/home/guest
```

In this example, user guest has specified the login shell as the C shell. The agent, therefore, runs the cmd1.ksh script using the environment defined in the $HOME/.login file.

Modify Resource Limits (UNIX)

When you define a Command job to run UNIX workload, you can set the job to modify resource limits on the agent computer for a given job. For example, you can define a job that modifies the maximum core file size and CPU times on the UNIX computer.

To modify resource limits for a Command job on UNIX

1. Define a Command job (see page 207).
2. Add the following attribute to the job definition:

   ulimit

   Specifies one or more resource types and their soft and hard limits. Specify the attribute as follows:

   ```
   ulimit: resource_type="soft_value,hard_value"
           [,resource_type="soft_value,hard_value"...]
   ```

3. Run the job.

   The resource limits on the UNIX computer are modified.

Notes:

- If the job runs under a non-root user ID, the hard limit will not be modified if the value in the job definition is greater than the hard limit on the UNIX computer.
- For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Modify Multiple Resource Limits

This example runs the procrun.sh script on the unixagent computer. The job modifies the following resource limits on the UNIX computer:

- The core file size limit is 100 KB (soft limit). The size can be increased to 200 KB (hard limit).
- The stack size limit is 250 KB (soft limit). The size can be increased to 300 KB (hard limit).
- The CPU time can be up to 4000 seconds.
- The process virtual size limit is 3332 KB (soft limit). The size can be increased to an unlimited value.

```
insert_job: cmd_job
job_type: CMD
machine: unixagent
command: /u1/procrun.sh
ulimit: c=”100,200”, s=”250,300”, t=”unlimited,4000”, m=”3332, unlimited”
```

Note: The resource limits are modified for the current job definition only. When you run another job, the default values will be used.

Customize the Run-time Environment for a Korn Shell Script (UNIX)

When you define a Command job to run a Korn shell script under a user's account, you can customize the job's run-time environment. The agent runs the job using the specified user's login environment and the run-time environment you specify.

To customize the run-time environment for a Korn shell script on UNIX

1. Define a Command job (see page 207).
2. Ensure that the oscomponent.loginshell parameter in the agent's agentparm.txt file is set to true.
3. Add the following attributes to the job definition:

   ```
   envvars: HOME=/directory_name
   envvars: ENV=/directory_name/myenv
   ```

4. Run the job.

   The run-time environment for the Korn shell script is customized. Before running the Korn script, the agent runs the /*directory_name*/.profile file and the /*directory_name*/myenv file.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Customize the Run-time Environment for a Bourne Shell Script (UNIX)

When you define a Command job to run a Bourne shell script under a user's account, you can customize the job's run-time environment. The agent runs the job using the specified user's login environment and the run-time environment you specify.

To customize the run-time environment for a Bourne shell script on UNIX

1. Define a Command job (see page 207).
2. Ensure that the oscomponent.loginshell parameter in the agent's agentparm.txt file is set to true.
3. Define a specific environment for the Korn shell.

4. Ask your agent administrator to configure the agent to run the Bourne shell script using the Korn shell
5. Run the job.

   The run-time environment for the Bourne shell script is customized.

Notes:

- The ENV variable only works for the Korn shell. The Korn shell is a superset of the Bourne shell. Anything that runs under the Bourne shell runs without modification under the Korn shell.
- For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Customize the Run-time Environment for a Bourne Shell Script

This example runs the cmd1.sh Bourne shell script. Before running the script, the agent runs the following:

- The login file for user guest
- The /esp/myenv file

```
insert_job: cmd_job
job_type: CMD
machine: unixagent
command: /home/bin/cmd1.sh
shell: /bin/ksh
owner: guest
envvars: ENV=/esp/myenv
```

Customize the Run-time Environment for a C Shell Script (UNIX)

When you define a Command job to run a C shell script under a user's account, you can customize the job's run-time environment. The agent runs the job using the specified user's login environment and the run-time environment you specify.

To customize the run-time environment for a C shell script on UNIX

1. Define a Command job (see page 207).
2. Ensure that the oscomponent.loginshell parameter in the agent's agentparm.txt file is set to true.
3. Set the environment in a .login file.

4. Add the following attribute to the job definition:

   `envvars: HOME=`*`login_file`*

   login_file

   Specifies the path to the .login file.

5. Run the job.

   The run-time environment for the C shell script is customized.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Customize the Run-time Environment for a C Shell Script

In this example, the agent picks up the environment for user root and runs the /home/bin/.login file before running the cmd1.csh C shell script. The agent is running as root. Note that the login shell for root is not the C shell.

```
insert_job: cmd_job
job_type: CMD
machine: unixagent
command: /home/guest/bin/cmd1.csh
envvars: HOME=/home/bin/
```

Define a Command Job to Run a Perl Script (UNIX)

You can define a Command job to run a Perl script on UNIX.

To define a Command job to run a Perl script

1. Ask your agent administrator to add the path of the Perl interpreter to the oscomponent.validshell parameter in the agent's agentparm.txt file, as shown in the following example:

   ```
   oscomponent.validshell=/usr/bin/sh,/bin/csh,/bin/ksh,/usr/local/bin/perl,/usr
   /local/bin/bash
   ```

   Note: If the oscomponent.checkvalidshell parameter is set to false, you do not need to perform the first step.

2. Define a Command job (see page 207).

3. Add the command attribute to the job definition using the following syntax:

   `command: /usr/`*`script`*

   script

   Specifies the path to the script you want to run.

   Example: cmd1.pl

4. Do *one* of the following:

   - Specify the path to the Perl interpreter in the first line of the script, as shown in the following example:

     ```
     Perl Script cmd1.pl
       #!/usr/local/bin/perl
       print " $0 @ARGV\n";
       while (( $var, $value) = each %ENV) {
         print "$var = $value\n";
       }
       $live=$ENV{pick};
       print " user variable pick = $live\n";
     ```

   - Specify the path to the Perl interpreter in the job definition, using the shell attribute, as shown in the following example:

     ```
     insert_job: perl_job
     job_type: CMD
     machine: unixagent
     command: /bin/cmd1.pl Hello world
     shell: /usr/local/bin/perl
     owner: guest
     ```

5. Run the job.

   The job runs the Perl script.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Run the Windows Command Interpreter (Windows)

You can schedule a job to run a command, a batch script, program executable using the Windows command interpreter (cmd.exe).

To run the Windows command interpreter

1. Verify the following parameters are set in the agent's agentparm.txt file:

   ```
   oscomponent.lookupcommand=true
   oscomponent.cmdprefix.force=true
   ```

2. Define a Command job (see page 207).
3. Add the command attribute to the job definition using the following syntax:

   ```
   command: command argument...
   ```

   command

   Specifies the cmd.exe command to run.

   Examples: copy, dir, echo

   argument...

   Specifies one or more arguments to pass to the cmd.exe command.

   Note: Separate each argument with a space. You must specify each argument in the order it is expected in the program.

4. Run the job.

   The job runs the specified cmd.exe command.

Notes:

- If the agent parameters specified in Step 1 are not set to true, you must explicitly invoke the command interpreter in the command attribute, as follows:

  ```
  command: "path\cmd.exe /C command argument..."
  ```

  path

  Specifies the path to cmd.exe. The path to cmd.exe depends on your Windows operating system version. For example, on Windows NT, the path is C:\WINNT\system32\cmd.exe.

- For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Run a Windows Operating System Command

This example lists the contents of the c:\temp directory. The oscomponent.lookupcommand and oscomponent.cmdprefix.force parameters in the agentparm.txt file are set to true, so the path to the command interpreter (*path*\cmd.exe /c) is automatically prefixed to the command before running the process

```
insert_job: cmd_job
job_type: CMD
machine: winprod
command: dir "c:\temp\"
```

Example: Explicitly Invoke the Command Interpreter

This example uses cmd.exe to copy a file to another location. The oscomponent.lookupcommand and oscomponent.cmdprefix.force parameters in the agentparm.txt file are not set to true, so the path to the command interpreter must be explicitly invoked in the command attribute.

```
insert_job: cmd_job
job_type: CMD
machine: winprod
command: "C:\Windows\system32\cmd.exe /C copy C:\env.txt C:\test\env.txt"
```

Access Network Resources (Windows)

When your agent runs as a Windows service, you can schedule Windows workload that accesses network resources. For example, you can specify UNC names and share names in your job definition.

Usually, when running a Windows program as a service, you are restricted to how you can access data on remote computers. For example, to access data on a remote computer as a specified user ID, you must run the Windows service with that user ID.

With the agent, however, those restrictions do not apply. Instead of running the agent service with a specific user ID, you can specify the user ID with the owner attribute in your job definition. To use the owner attribute, your agent must run as a Windows service under the local system account (the default configuration).

To access Windows network resources

1. Verify with your agent administrator that the agent is running as a Windows service under the local system account.
2. Ask your scheduling manager administrator to define a user ID and password on the scheduling manager that has access to the file on the remote Windows system.
3. Define a Command job (see page 207).

4. Add the following attributes to the job definition:

   command

   Specifies a command, executable, application, or batch file to run when all the starting conditions are met.

   Notes:

   - You can specify UNC (Universal Naming Convention) names. A UNC name is the name of a file or other resource that begins with two backslashes (\\), indicating that it exists on a remote computer.
   - You can specify share names. A share name is an alias for the path the resource exists in.
   - You can specify the share names C$ and ADMIN$ if the agent service logs on to a remote Windows server as a user with administrative authority. The agent can then access remote resources that are not marked as shared.

   owner

   Specifies the Windows user ID and the domain the user ID belongs to.

   Default: The default owner (the user ID who invokes jil to define the job)

5. Run the job.

   The job accesses the specified Windows network resources.

Notes:

- For more information about attributes and their JIL syntax, see the *Reference Guide*.
- Before accessing network resources with your agent, verify that you are complying with the terms of your agent license agreement. In most situations, you are permitted to access data on remote computers; however, scripts or executable files run by an agent should use the CPU and memory of the computer where the agent resides.
- Although not recommended, your agent administrator can run the agent as a Windows service under a local user account (the This Account option). When you run the service under a local user account, when the service starts, it runs using the security context of the specified user account. If the user account and password are valid, the service process has access to network resources.
- When you access a remote computer from an agent on Windows, the user ID defined in the owner attribute or in the This Account option is a domain user. If the local and remote servers are standalone servers, you must have the same user IDs and passwords defined on both servers.
- For more information on configuring and running the agent as a Windows service, see the *CA Workload Automation Agent for UNIX, Linux, or Windows Implementation Guide*.

Example: Run a Command on a Remote Server

In this example, the path c:\WINNT\Profiles\Visitor\Desktop\ has the share name MyDesktop. The command notify.cmd is in that path on the CYBNT server. JDOE is a user ID in the CYBDOM domain and has access to the notify.cmd command. JDOE's password is defined on the scheduling manager.

```
insert_job: exe_job1
job_type: CMD
machine: NT20
command: \\CYBNT\MyDesktop\notify.cmd
owner: CYBDOM\JDOE
```

Example: Run an Executable in Public Folder on a Remote Server

This example runs calc.exe on the CYBNT server. CYBUSER is a user ID in the CYBDOM domain. CYBUSER is defined on the scheduling manager and has access permission to the public folder.

```
insert_job: exe_job2
job_type: CMD
machine: NT30
command: \\CYBNT\public\calc.exe
owner: CYBDOM\CYBUSER
```

Example: Access a Remote Resource Using the C$ Share Name

In this example, drive C is accessed by an administrator over the network through an agent. The agent is running under the System Account option. The agent runs the test application in the c:\working directory on the server CYBNT. The directory c:\working is not a shared resource. The user admin1 is a valid user on both the local and remote computers and belongs to the Administrators group. admin1 is also in the CYBDOM domain.

```
insert_job: exe_job3
job_type: CMD
machine: NT30
command: \\CYBNT\C$\working\test
owner: CYBDOM\admin1
```

Specifying a Password for a User ID (Windows)

You can define a Command job to access Windows network resources by specifying the user ID and the domain name the user ID belongs to using the owner attribute. The resources are accessed under the specified user ID.

Note: To use the owner attribute in a Command job that runs on Windows, your agent must run as a Windows service under the local system account.

If the user ID requires a password, your administrator must define the password on the scheduling manager. For security reasons, you do not define the password in the job definition. Your administrator must define and store the password in the CA Workload Automation AE database using the autosys_secure command.

When you specify a user ID that requires a password in a job definition, the scheduling manager sends the agent the user ID and password pair (the password is encrypted). The scheduling manager searches the repository for an entry matching the specified owner. The results from the search, the encrypted password or null, are provided to the agent to run the job.

The following job types require a user password:

- Database
- FTP
- PeopleSoft
- SAP
- Command (for Windows)

Chapter 8: Database Jobs

This section contains the following topics:

Database Jobs

Database jobs let you automate common database tasks on Oracle, Microsoft SQL Server, and IBM DB2 databases.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Databases.

You can define the following database jobs:

SQL

Lets you run an SQL query against a database.

Database Stored Procedure

Lets you run a stored procedure.

Database Trigger

Lets you monitor for added, deleted, and updated rows in a database table.

Database Monitor

Lets you monitor for added and deleted rows in a database table.

How Database Trigger Jobs Differ from Database Monitor Jobs

You can monitor database changes either by using a Database Trigger job or a Database Monitor job. The Database Trigger job offers the following advantages over Database Monitor jobs:

- You can monitor for more conditions. With Database Trigger jobs, you can monitor for rows added, deleted, and updated. With Database Monitor jobs, you can only monitor for rows added and deleted. Database Trigger jobs define triggers to the underlying database being monitored. Database Monitor jobs poll the database table for changes in row counts or column values.
- Database Trigger jobs detect all changes made to the database; Database Monitor jobs monitor for changes only in 10 second intervals, by default.

Suppose that you want to send a notification when a new row is added. Within a 10-second interval, assume a row is added while another row is deleted. A Database Trigger job that monitors for an INSERT would complete and send the notification when the new row is detected. A Database Monitor job that monitors for an INCREASE would not complete or send a notification because no change in the total number of rows was detected.

Each Database Trigger job creates a database trigger on the database. The database trigger templates are provided with the agent. Before using a Database Trigger job, consult with your database administrator.

Notes:

- A table being monitored should not be dropped because the Database Trigger or Database Monitor job remains in the RUNNING status even if the table has been dropped.
- For more information about the database trigger templates, see the db.trig.propfile parameter in the *CA Workload Automation Agent for Databases Implementation Guide*.

User IDs and Passwords for Database Jobs

All database jobs require a user ID that has the appropriate permissions to access the information in the database. The job runs under that user ID. You specify a user ID using the owner attribute.

These database user IDs and passwords must be defined on CA Workload Automation AE by using the autosys_secure command. When you define a database job, specify a database user ID using the owner attribute, or use the default owner value.

Note: For more information about the autosys_secure command, see the *Reference Guide*.

Define a Database Monitor Job

You can define a Database Monitor job to monitor a database table for an increase or decrease in the number of rows. To monitor the database table for specific changes, you can add a monitor condition to the job definition. When the condition is met, the job completes.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Databases.

To define a Database Monitor job

1. Insert a job and specify the following attributes in the definition:

   job_type: DBMON

   Specifies that the job type is Database Monitor.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   tablename

   Specifies the name of the database table to monitor for the changes.

2. Do *one* of the following:

   - Ensure that a default database resource location is defined in the agent's agentparm.txt file using the db.default.url parameter.
   - Add the following attribute to the definition:

     connect_string

     Specifies the database resource location.

     Note: This attribute overrides the db.default.url agent parameter.

3. (Optional) Specify the following attribute:

   owner

   Specifies the user ID that the job runs under. This value overrides the default owner of the job.

   Default: The user ID who invokes jil to define the job

   Note: Windows authentication is not supported.

4. (Optional) Specify optional Database Monitor attributes:
   - continuous
   - job_class
   - job_terminator
   - monitor_cond
   - monitor_type
   - user_role
5. (Optional) Specify common attributes that apply to all job types.

   The Database Monitor job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor a Table for Added Rows

This example monitors for an increase in the number of rows in the staff table. The table is in a SQL Server database named ORDERS.

```
insert_job: dbmon_add
job_type: DBMON
machine: dbagent
owner: dbuser@MSSQL
tablename: staff
monitor_type: INCREASE
connect_string: "jdbc:sqlserver://myhost:1433;DatabaseName=ORDERS"
```

Example: Monitor a Table for Added or Deleted Rows

This example monitors the STAFF table for a change in the number of rows. When a row that contains the name Jonson is added or deleted, the job completes. The job connects to the default database resource location defined on the agent.

```
insert_job: dbmon_job
job_type: DBMON
machine: dbagent
owner: entadm@myhost
monitor_type: VARIANCE
tablename: staff
monitor_cond: NAME='Jonson'
```

Example: Monitor a Table for Added Rows With a Condition

This example monitors the emp table for an increase in the number of rows. When a new row has a sal greater than 100000, the job completes. The job connects to a SQL Server database named ORDERS. The database user ID is set to the user who invokes jil to define the job (the default owner).

```
insert_job: dbmon1
job_type: DBMON
machine: DB_agent
owner: dbuser@MSSQL
monitor_type: INCREASE
monitor_cond: sal>100000
tablename: emp
connect_string: "jdbc:sqlserver://myhost:1433;DatabaseName=ORDERS"
```

More information:

Insert a Job (see page 87)

Define a Database Trigger Job

You can define a Database Trigger job to monitor a database table for added, deleted, or updated rows. To monitor the database table for specific changes, you can add a condition to the job definition. When the condition is met, the job completes.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Databases.

To define a Database Trigger job

1. Insert a job and specify the following attributes in the definition:

   job_type: DBTRIG

   Specifies that the job type is Database Trigger.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   dbtype

   Specifies the type of the database that the job monitors.

   tablename

   Specifies the name of the database table to monitor for the changes.

2. Do *one* of the following:

   - Ensure that a default database resource location is defined in the agent's agentparm.txt file using the db.default.url parameter.
   - Add the following attribute to the definition:

     connect_string

     Specifies the database resource location.

     Note: This attribute overrides the db.default.url agent parameter.

3. (Optional) Specify the following attribute:

   owner

   Specifies the user ID that the job runs under. This value overrides the default owner of the job.

   Default: The user ID who invokes jil to define the job

   Note: This user ID must be authorized to create triggers on the database or schema the table belongs to. For Microsoft SQL Server, this user ID must own the database table identified by the tablename attribute. The password for the user must be defined in the database using the autosys_secure command. Windows authentication is not supported.

4. (Optional) Specify optional Database Trigger attributes:
   - continuous
   - job_class
   - job_terminator
   - trigger_cond
   - trigger_type
   - user_role
5. (Optional) Specify common attributes that apply to all job types.

   The Database Trigger job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor a Table for Added Rows

This example monitors the emp table. The trigger_type attribute is not specified, so the job monitors for added rows by default. When a row is added, the job completes. The database resource location is defined on the agent, so the connect_string attribute is not required in the job definition.

```
insert_job: dbtrig_job
job_type: DBTRIG
machine: DB_agent
owner: dbuser@ORA
dbtype: Oracle
tablename: emp
```

More information:

Insert a Job (see page 87)

Examples: Monitoring Oracle Database Tables

The following examples are Database Trigger Jobs that monitor Oracle database tables:

Note: These examples use optional database attributes. For more information about the optional attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor an Oracle Database Table for Deleted Rows

This example monitors the emp table for deleted rows. The job runs under the user named scott, who has the authority to create triggers on the database or schema the table belongs to. When a row is deleted, the job completes.

```
insert_job: dbtrig2
job_type: DBTRIG
machine: DB_agent
dbtype: Oracle
trigger_type: DELETE
tablename: emp
owner: scott@orcl
connect_string: "jdbc:oracle:thin:@myhost:1521:orcl"
```

Example: Monitor an Oracle Database Table for an Added or Deleted Row

This example monitors the emp table for an added row or a deleted row. The job runs under the user named scott, who has the authority to create triggers on the database or schema the table belongs to. The job remains in a RUNNING state while waiting for an added or deleted row. When a row is either added or deleted, the job completes.

```
insert_job: dbtrig_ora
job_type: DBTRIG
machine: dbagent
dbtype: Oracle
trigger_type: DELETE,INSERT
tablename: emp
owner: scott@orcl
connect_string:"jdbc:oracle:thin:@myhost:1521:orcl"
```

Examples: Specify Trigger Conditions for Deleted Rows in an Oracle Database

- This example monitors the emp table for deleted rows. The job runs under a user who has the authority to create triggers on the database or schema the table belongs to. The job connects to the default database resource location defined on the agent. When a row containing deptno 75 is deleted, the job completes.

```
insert_job: dbtrig_delete
job_type: DBTRIG
machine: dbagent
dbtype: Oracle
trigger_type: DELETE
trigger_cond: old.deptno=75
tablename: emp
owner: scott@orcl
```

- This example monitors the emp table for added rows. The job runs under a user who has the authority to create triggers on the database or schema the table belongs to. When a row containing an ename beginning with the letter g is added, the job completes.

```
insert_job: dbtrig_insert
job_type: DBTRIG
machine: dbagent
dbtype: Oracle
trigger_type: INSERT
trigger_cond: new.ename like 'g%%'
tablename: emp
owner: scott@orcl
connect_string:"jdbc:oracle:thin:@myhost:1521:orcl"
```

- This example monitors the emp table for added or updated rows. The job runs under a user who has the authority to create triggers on the database or schema the table belongs to. The job completes when a new or updated row does not contain a job field equal to sales.

  Note: The <> symbol indicates not equal to.

```
insert_job: dbtrig_insertupdate
job_type: DBTRIG
machine: dbagent
dbtype: Oracle
trigger_type: INSERT,UPDATE
trigger_cond: new.job<>'sales'
tablename: emp
owner: scott@orcl
connect_string:"jdbc:oracle:thin:@myhost:1521:orcl"
```

Examples: Monitoring Microsoft SQL Server Database Tables

The following examples are Database Trigger jobs that monitor Microsoft SQL Server database tables:

Note: These examples use optional database attributes. For more information about the optional attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor a SQL Server Database Table for a New or Deleted Row

This example monitors the stores table for an added row or a deleted row. The job runs under the sa user, who owns the table and is authorized to create triggers on the database or schema the table belongs to. The job remains in a RUNNING state waiting for an added or deleted row. When a row is either added or deleted, the job completes.

```
insert_job: dbtrig1
job_type: DBTRIG
machine: DB_agent
trigger_type: DELETE,INSERT
tablename: stores
dbtype: MSSQL
owner: sa@myhost
connect_string:"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
```

Example: Monitor a SQL Server Database Table for Two Changes

This example monitors the sales table for changes to the ord_date and qty columns. The job runs under the sa user, who owns the table and is authorized to create triggers on the database or schema the table belongs to. The job completes only when both columns have changed.

```
insert_job: dbtrig_sqlsvr
job_type: DBTRIG
machine: dbagent
dbtype: MSSQL
owner: sa@myhost
connect_string: "jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
tablename: sales
trigger_type: UPDATE
trigger_cond: UPDATE(ord_date) and UPDATE(qty)
```

Example: Monitor a SQL Server Database Table for Added Rows with a Trigger Condition

This example monitors the sales table for added rows. The job runs under the sa user, who owns the table and is authorized to create triggers on the database or schema the table belongs to. The job connects to the default database resource location defined on the agent. When the qty for inserted title ID TC7777 is greater than or equal to 20, the job completes.

```
insert_job: dbtrig3
job_type: DBTRIG
machine: DB_agent
dbtype: MSSQL
trigger_type: INSERT
trigger_cond: (select QTY from INSERTED where TITLE_ID='TC7777')>=20
tablename: sales
owner: sa@myhost
```

Example: Monitor a SQL Server Database Table for Deleted Rows

This example monitors the sales table for deleted rows. The job runs under the sa user, who owns the table and is authorized to create triggers on the database or schema the table belongs to. The job completes when a row is deleted.

```
insert_job: dbtrig4
job_type: DBTRIG
machine: DB_agent
dbtype: MSSQL
trigger_type: DELETE
tablename: sales
owner: sa@myhost
connect_string:"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
```

Examples: Monitoring IBM DB2 Database Tables

The following examples are Database Trigger jobs that monitor IBM DB2 Server database tables:

Note: These examples use optional database attributes. For more information about the optional attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor an IBM DB2 Database Table for Added Rows with a Trigger Condition

This example monitors the STAFF table for added rows. The job runs under the entadm user, who owns the table and is authorized to create triggers on the database or schema the table belongs to. When the total number of rows is greater than or equal to 37, the job completes.

```
insert_job: dbtrig1
job_type: DBTRIG
machine: DB_agent
dbtype: DB2
trigger_type: INSERT
trigger_cond: (select count(*) from STAFF)>=37
tablename: STAFF
owner: entadm@myhost
connect_string:"jdbc:db2://172.31.255.255:50000/SAMPLE"
```

Example: Monitor an IBM DB2 Database Table for Changes

This example monitors the STAFF table for changes. The job connects to the SAMPLE database and runs under the entadm user, who is authorized to create triggers on the database or schema the table belongs to. The job completes when the table has changed.

```
insert_job: dbtrig_db2
job_type: DBTRIG
machine: dbagent
dbtype: DB2
owner: entadm@myhost
connect_string:"jdbc:db2://172.31.255.255:50000/SAMPLE"
tablename: STAFF
dbtype: DB2
trigger_type: UPDATE
```

Example: Monitoring a Sybase Database Table

The following example is a Database Trigger Job that monitors a Sybase database table:

Note: These examples use optional database attributes. For more information about the optional attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor a Sybase Database Table for Changes

This example monitors the ap_invoices table for changes. The job connects to the Sybase database named APDB that is reachable on port 5001 on a host named myhost. The job completes when the table has changed.

```
insert_job: db_sqltrig_upd
job_type: DBTRIG
machine: localhost
dbtype: Sybase
connect_string: "jdbc:sybase:Tds:myhost:5001/APDB"
trigger_type: UPDATE
tablename: ap_invoices
owner: admin@myhost
```

Define a Database Stored Procedure Job

You can define a Database Stored Procedure job to invoke a procedure stored in a database. You can add criteria to the job definition to test the procedure's output. If the result matches the criteria, the job completes successfully. When the procedure executes, the output parameter values from the database are returned to CA Workload Automation AE. You can view the output parameter values in the job's spool file. By default, the agent separates the output parameter values in the return string with a vertical bar (|).

If you are using Oracle or SQL Server, you can also define a Database Stored Procedure job to run a stored function.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Databases.

To define a Database Stored Procedure job

1. Insert a job and specify the following attributes in the definition:

   job_type: DBPROC

   Specifies that the job type is Database Stored Procedure.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   sp_name

   Specifies the database stored procedure to run.

2. Do *one* of the following:

   - Ensure that a default database resource location is defined in the agent's agentparm.txt file using the db.default.url parameter.
   - Add the following attribute to the definition:

     connect_string

     Specifies the database resource location.

     Note: This attribute overrides the db.default.url agent parameter.

3. (Optional) Specify the following attribute:

   owner

   Specifies the user ID that the job runs under. This value overrides the default owner of the job.

   Default: The user ID who invokes jil to define the job

   Note: Windows authentication is not supported.

4. (Optional) Specify optional Database Stored Procedure attributes:
   - job_class
   - job_terminator
   - result_type
   - sp_arg
   - success_criteria
   - user_role
5. (Optional) Specify common attributes that apply to all job types.

   The Database Stored Procedure job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Invoke a Stored Procedure from a Database

This example invokes the calcproc stored procedure. The default database resource location is defined on the agent, so the connect_string attribute is not required in the job definition.

```
insert_job: sp_default
job_type: DBPROC
machine: DB_agent
owner: dbuser@dbhost
sp_name: calcproc
```

Example: Invoke a Stored Procedure from a SQL Server Database

This example invokes the byroyalty stored procedure located in the pubs database. When the job runs, a value of 40 is passed to the input parameter named percentage.

```
insert_job: sp1_job
job_type: DBPROC
machine: DB_agent
owner: sa
sp_name: byroyalty
sp_arg: ignore=yes, name=percentage, argtype=IN, datatype=INTEGER, value=40
connect_string:"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
```

Example: Invoke a Stored Procedure with Input and Output Parameters from a SQL Server Database

This example invokes the following stored procedure located in the pubs database. The procedure returns a value from the emp table.

```
CREATE PROCEDURE EMPLOY
(@f_name VARCHAR(20),
@l_name VARCHAR(30),
@pubid CHAR(4) OUTPUT)
AS BEGIN
SELECT
@pubid=pub_id
FROM emp
WHERE
fname=@f_name
and
lname=@l_name
print @l_name+@f_name+@pubid
END
GO
```

The job returns the pubid that matches the employee named John Doe. The pubid is recorded in the job's spool file.

```
insert_job: sp2_job
job_type: DBPROC
machine: DB_agent
sp_name: EMPLOY
sp_arg: ignore=yes, name=f_name, argtype=IN, datatype=VARCHAR, value=John
sp_arg: ignore=yes, name=l_name, argtype=IN, datatype=VARCHAR, value=Doe
sp_arg: ignore=no, name=pubid, argtype=OUT, datatype=CHAR
owner: sa
connect_string:"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
```

Example: Invoke a Stored Procedure with Input and Output Parameters from an IBM DB2 Database

This example invokes the following stored procedure under the user entadm:

```
CREATE PROCEDURE DEPT_MEDIAN
  (IN deptNumber SMALLINT, OUT medianSalary DOUBLE)
  LANGUAGE SQL
  BEGIN
    DECLARE v_numRecords INTEGER DEFAULT 1;
    DECLARE v_counter INTEGER DEFAULT 0;
    DECLARE c1 CURSOR FOR
      SELECT CAST(salary AS DOUBLE) FROM staff
        WHERE DEPT = deptNumber
        ORDER BY salary;
      DECLARE EXIT HANDLER FOR NOT FOUND
        SET medianSalary = 6666;
-- initialize OUT parameter
        SET medianSalary = 0;
        SELECT COUNT(*) INTO v_numRecords FROM staff
          WHERE DEPT = deptNumber;
        OPEN c1;
        WHILE v_counter < (v_numRecords / 2 + 1) DO
          FETCH c1 INTO medianSalary;
          SET v_counter = v_counter + 1;
        END WHILE;
        CLOSE c1;
  END
```

DEPT_MEDIAN returns the median salary for the department with deptNumber 20 from the STAFF table. The median salary, 18171.25, is recorded in the job’s spool file.

```
insert_job: deptmed
job_type: DBPROC
machine: DB_agent
sp_name: ENTADM.DEPT_MEDIAN
sp_arg: ignore=yes, name=deptNumber, argtype=IN, datatype=SMALLINT, value=20
sp_arg: ignore=no, name=medianSalary, argtype=OUT, datatype=DOUBLE
owner: entadm
connect_string:"jdbc:db2://172.31.255.255:50000/SAMPLE"
```

The spool file for this job contains the following output:

```
----------------------------------------------------------------
Output of messages for workload object DEPTMED/DBAPPL.7/MAIN
Start date Thu Aug 31 15:23:44 EDT 2006
----------------------------------------------------------------
{ call ENTADM.DEPT_MEDIAN(?, ?) }
medianSalary=18171.25
```

More information:

Insert a Job (see page 87)

Define an SQL Job

You can define an SQL job to run an SQL query against an Oracle, SQL Server, Sybase, or DB2 database. When the job runs, the SQL statement is invoked and the results are stored in an output file or job spool file. You can also add criteria to the job definition to test the query result. If the result matches the criteria, the job completes successfully. Otherwise, the job fails.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Databases.

To define an SQL job

1. Insert a job and specify the following attributes in the definition:

   job_type: SQL

   Specifies that the job type is SQL.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   sql_command

   Specifies the SQL statement to run against a database table.

2. Do *one* of the following:

   - Ensure that a default database resource location is defined in the agent's agentparm.txt file using the db.default.url parameter.
   - Add the following attribute to the definition:

     connect_string

     Specifies the database resource location.

     Note: This attribute overrides the db.default.url agent parameter.

3. (Optional) Specify the following attribute:

   owner

   Specifies the user ID that the job runs under. This value overrides the default owner of the job.

   Default: The user ID who invokes jil to define the job

   Note: Windows authentication is not supported.

4. (Optional) Specify optional SQL attributes:
   - destination_file
   - job_class
   - job_terminator
   - success_criteria
   - user_role
5. (Optional) Specify common attributes that apply to all job types.

   The SQL job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Delete a Row from a Table

This example deletes the row for stor_id 6523 from the stores table. The default database resource location is defined on the agent, so the connect_string attribute is not required in the job definition.

```
insert_job: deletejob
job_type: SQL
machine: DB_agent
sql_command: DELETE FROM stores WHERE stor_id='6523'
owner: scott@orcl
```

More information:

Insert a Job (see page 87)

Examples: Running SQL Queries Against Oracle Database Tables

The following examples are jobs that run SQL queries against Oracle database tables:

Note: These examples use optional database attributes. For more information about the optional attributes and their JIL syntax, see the *Reference Guide*.

Example: Add a Row to an Oracle Database Table

This example adds a row of data to the emp table.

```
insert_job: insertjob
job_type: SQL
machine: DB_agent
sql_command: INSERT INTO EMP(EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
VALUES(2476, 'robert', 'sales', 435, '01-OCT-2011', 65000, 10, 75)
owner: scott@orcl
connect_string:"jdbc:oracle:thin:@myhost:1521:orcl"
```

Example: Update a Row in an Oracle Database Table

This example updates a record in the emp table and changes the sal to 75,000 for the employee with ename robert.

```
insert_job: updatejob
job_type: SQL
machine: DB_agent
sql_command: UPDATE EMP SET SAL=75000 where ENAME='robert'
owner: scott@orcl
connect_string:"jdbc:oracle:thin:@myhost:1521:orcl"
```

Example: Delete a Row from an Oracle Database Table

This example deletes a row from the emp table for the employee with ename robert.

```
insert_job: deletejob
job_type: SQL
machine: DB_agent
sql_command: DELETE FROM EMP WHERE ENAME='robert'
owner: scott@orcl
connect_string:"jdbc:oracle:thin:@myhost:1521:orcl"
```

Example: Return Data from an Oracle Database Table that Match a Condition

This example queries the emp table for enames that have salaries greater than 40,000. If the query returns an ename that begins with the letter d, the job completes:

```
insert_job: selectjob
job_type: SQL
machine: DB_agent
sql_command: SELECT ename FROM emp WHERE sal > 40000
owner: scott@orcl
success_criteria: ENAME=d.*
connect_string:"jdbc:oracle:thin:@myhost:1521:orcl"
destination_file: /emp/salary.txt
```

For example, the salary.txt file contains the following output:

```
Output for: SELECT ename FROM emp WHERE sal > 40000
ENAME
-----------
donald
```

Examples: Running SQL Queries Against Microsoft SQL Server Database Tables

The following examples are jobs that run SQL queries against Microsoft SQL Server database tables:

Note: These examples use optional database attributes. For more information about the optional attributes and their JIL syntax, see the *Reference Guide*.

Example: Add a Row to a SQL Server Database Table

This example adds a row for a new store to the stores table.

```
insert_job: insertjob
job_type: SQL
machine: DB_agent
sql_command: INSERT INTO stores(stor_id, stor_name, stor_address, city, state, zip)
VALUES('6523', 'BooksMart', '6523 Main St.', 'San Diego', 'CA', '93223')
owner: sa@myhost
connect_string:"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
```

Example: Delete a Row from a SQL Server Database Table

This example deletes the row for stor_id 6523 from the stores table.

```
insert_job: deletejob
job_type: SQL
machine: DB_agent
sql_command: DELETE FROM stores WHERE stor_id='6523'
owner: sa@myhost
connect_string:"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
```

Example: Update a Row in a SQL Server Database Table

This example updates the row in the sales table that matches ord_num 6871 and changes the values for the ord_date and qty.

```
insert_job: updatejob
job_type: SQL
machine: DB_agent
sql_command: UPDATE sales SET ord_date='6/15/2006', qty=10 WHERE ord_num='6871'
owner: sa@myhost
connect_string:"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
```

Example: Return Data from a SQL Server Database Table that Match a Condition

This example queries the sales table for ord_num that have a qty greater than 20. The ord_num that match the query appear in the output file ordnum.txt.

```
insert_job: selectjob
job_type: SQL
machine: DB_agent
sql_command: SELECT ord_num FROM sales WHERE qty > 20
owner: sa@myhost
success_criteria: ord_num=A2976
connect_string:"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
destination_file: "C:\sales\ordnum.txt"
```

The ordnum.txt file contains the following ord_num:

```
A2976
QA7442.3
P2121
N914014
P3087a
P3087a
X999
P723
QA879.1
```

The job completes because the query returns an ord_num that matches the job criteria A2976.

Suppose that we change the success_criteria attribute to the following:

```
success_criteria: B+[0-9]
```

In this case, the query would return the same order numbers, but the job fails because it cannot find a matching ord_num containing the letter B and followed by a number.

Examples: Running SQL Queries Against Sybase Database Tables

The following examples are jobs that run SQL queries against Sybase database tables:

Note: These examples use optional database attributes. For more information about the optional attributes and their JIL syntax, see the *Reference Guide*.

Example: Add a Row to a Sybase Database Table

This example adds a row for a new store to the stores table.

```
insert_job: insertjob
job_type: SQL
machine: DB_agent
sql_command: INSERT INTO stores(stor_id, stor_name, stor_address, city, state, zip)
VALUES('6523', 'BooksMart', '6523 Main St.', 'San Diego', 'CA', '93223')
owner: sa@myhost
connect_string:"jdbc:sybase:Tds:myhost:5001/APDB"
```

Example: Delete a Row from a Sybase Table

This example deletes the row for stor_id 6523 from the stores table.

```
insert_job: deletejob
job_type: SQL
machine: DB_agent
sql_command: DELETE FROM stores WHERE stor_id='6523'
owner: sa@myhost
connect_string:"jdbc:sybase:Tds:myhost:5001/APDB"
```

Example: Update a Row in a Sybase Table

This example updates the row in the sales table that matches ord_num 6871 and changes the values for the ord_date and qty.

```
insert_job: updatejob
job_type: SQL
machine: DB_agent
sql_command: UPDATE sales SET ord_date='6/15/2006', qty=10 WHERE ord_num='6871'
owner: sa@myhost
connect_string:"jdbc:sybase:Tds:myhost:5001/APDB"
```

Examples: Running SQL Queries Against IBM DB2 Database Tables

The following examples are jobs that run SQL queries against IBM DB2 database tables:

Note: These examples use optional database attributes. For more information about the optional attributes and their JIL syntax, see the *Reference Guide*.

Example: Add a Row to an IBM DB2 Database Table

This example adds a row of data to the STAFF table under the user entadm.

```
insert_job: insertjob
job_type: SQL
machine: DB_agent
sql_command: INSERT into ENTADM.STAFF(ID, NAME, DEPT, JOB, YEARS, SALARY, COMM)
VALUES(556, 'Jonson', 84, 'Sales', 1, 40500.50, 100)
owner: entadm@myhost
connect_string:"jdbc:db2://172.31.255.255:50000/SAMPLE"
```

Example: Update a Row in an IBM DB2 Database Table

This example updates a record in the STAFF table under the user entadm. The job changes the years to 3 for the employee with the name Jonson.

```
insert_job: updatejob
job_type: SQL
machine: DB_agent
sql_command: UPDATE ENTADM.STAFF SET YEARS=3 where NAME="Jonson"
owner: entadm@myhost
connect_string:"jdbc:db2://172.31.255.255:50000/SAMPLE"
```

Example: Delete a Row from an IBM DB2 Database Table

This example deletes a row from the STAFF table under the user entadm for the employee with the name Jonson.

```
insert_job: deletejob
job_type: SQL
machine: DB_agent
sql_command: DELETE FROM ENTADM.STAFF where NAME="Jonson"
owner: entadm@myhost
connect_string:"jdbc:db2://172.31.255.255:50000/SAMPLE"
```

Example: Return Data from an IBM DB2 Database Table that Match a Condition

This example queries the STAFF table under the user entadm for names that have salaries greater than 40,000. If the query returns a name that begins with the letter J, the job completes.

```
insert_job: selectjob
job_type: SQL
machine: DB_agent
sql_command: SELECT NAME FROM ENTADM.STAFF where SALARY > 40000
owner: entadm@myhost
success_criteria: NAME=J.*
connect_string:"jdbc:db2://172.31.255.255:50000/SAMPLE"
destination_file: /staff/salary.txt
```

For example, the salary.txt file contains the following output:

```
Output for: SELECT NAME FROM ENTADM.STAFF where SALARY > 40000
NAME
-----------
Jonson
```

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following Database job attributes have default values:

connect_string

Specifies the database resource location.

Default: db.default.url agent parameter, if specified

Note: If you do not specify the connect_string attribute, the default database resource location must be defined in the db.default.url parameter in the agent's agentparm.txt file. Otherwise, the job fails.

destination_file (SQL jobs only)

Specifies the output destination file that stores the SQL query results.

Default: spooldir agent parameter, if specified

monitor_type (Database Monitor jobs only)

Specifies the type of database change to monitor for.

Default: VARIANCE (The job monitors for an increase or a decrease in the number of rows in the table.)

owner

Specifies the user ID that the job runs under.

Default: The user ID who invokes jil to define the job

Note: Windows authentication is not supported.

trigger_type (Database Trigger jobs only)

Specifies the type of database change to monitor for.

Default: INSERT (The job monitors for an insertion of a row in the table.)

user_role

Specifies the Oracle database user type.

Default: db.default.userType agent parameter, if specified

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Override Default Values

Suppose that you want to run an SQL job that queries the NEWORDS table. This job overrides the default database resource location defined on the agent using the connect_string attribute. This job also overrides the default owner with the dbuser1 user ID, who is logged in with sysdba privileges. The output is stored in the job's spool file by default.

```
insert_job: QRY1
job_type: SQL
machine: dbagent
owner: dbuser1@myhost
user_role: as sysdba
sql_command: SELECT * from NEWORDS
connect_string: "jdbc:oracle:thin:@172.31.255.255:1433:ORDERS"
```

Chapter 9: File Trigger Jobs

This section contains the following topics:

File Trigger Jobs

File Trigger jobs let you monitor file activity. You can define File Trigger jobs for UNIX, Linux, Windows, or i5/OS systems.

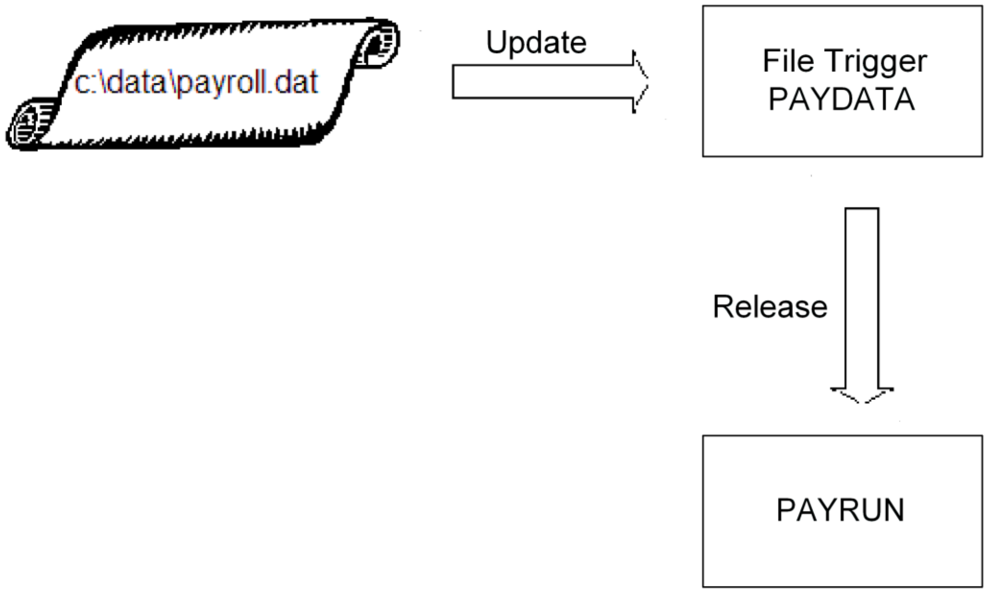
Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

The File Trigger job can monitor when a file is created, updated, deleted, expanded, or shrunk, and when a file exists or does not exist.

Example: Monitor for an Update to a File

Suppose that a File Trigger job named PAYDATA monitors for an update to the payroll.dat file on a Windows computer. When the file is updated, the job completes and the scheduling manager releases a job named PAYRUN.

The following diagram shows the scenario:



Define a File Trigger Job

You can define a File Trigger job to monitor when a file is created, updated, deleted, expanded, or shrunk, and when a file exists or does not exist. By default, File Trigger jobs monitor for the existence of a file.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows.

To define a File Trigger job

1. Insert a job and specify the following attributes in the definition:

   job_type: FT

      Specifies that the job type is File Trigger.

   machine

      Specifies the name of an existing machine definition where the agent runs.

   watch_file

      Specifies the path to and name of one or more files to monitor.

2. (Optional) Specify optional File Trigger attributes:
   - continuous
   - job_class
   - job_terminator
   - watch_file_change_type
   - watch_file_change_value
   - watch_file_groupname
   - watch_file_owner
   - watch_file_recursive
   - watch_file_type
   - watch_file_win_user
   - watch_no_change
3. (Optional) Specify common attributes that apply to all job types.

   The File Trigger job is defined. When the job runs, it monitors for the existence of the specified file.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor for the Existence of a File

This example monitors for the existence of the batch.input file in the tmp directory. If the file exists in the directory, the job completes. If the file does not exist in that directory, the job fails.

```
insert_job: ft_job
job_type: FT
machine: unixagt
watch_file: /tmp/batch.input
watch_file_type: EXIST
```

More information:

Insert a Job (see page 87)

Monitor for Other Types of File Activity

A File Trigger job checks for the existence of a file by default. You can define the job to monitor for other types of file activity.

To monitor for another type of file activity

1. Define a File Trigger job (see page 266).
2. Add the following attribute to the job definition:

   watch_file_type

   Specifies the type of file activity that a File Trigger job monitors for. Options are the following:

   - CREATE—Indicates that the file trigger occurs when the file is created. This is the default.
   - DELETE—Indicates that the file trigger occurs when the file is deleted.
   - EXIST—Indicates that the file trigger occurs if the file exists. If the file does not exist, the job fails.
   - EXPAND—Indicates that the file trigger occurs when the file size increases.
   - NOTEXIST—Indicates that the file trigger occurs if the file does not exist. If the file exists, the job fails.
   - SHRINK—Indicates that the file trigger occurs when the file size decreases.
   - UPDATE—Indicates that the file trigger occurs when the file is updated.
   - GENERATE—Indicates that the file trigger occurs when the file remains unchanged for the amount of time specified in the watch_no_change attribute.

3. Add the following attributes to the job definition if you specified CREATE, EXPAND, or SHRINK in the watch_file_type attribute:

   watch_file_change_type

   Specifies the type of change to detect when monitoring a file for an increase or decrease in size. Options are the following:

   - DELTA—Specifies that the job monitors for a change in the file size.
   - PERCENT—Specifies that the job monitors for a change in the file size by percentage.
   - SIZE—Specifies that the job monitors the file to reach a specified size. This is the default.

   watch_file_change_value

   Specifies the amount of change in bytes or the limit in size to monitor for.

4. Run the job.

   The job monitors the file for the specified activity.

Notes:

- For more information about attributes and their JIL syntax, see the *Reference Guide*.
- By default, the agent scans for the monitored conditions every 30 seconds. If the file changes more than once between scans, the trigger occurs only once or not at all. For example, suppose that your job monitors for the creation of a file and that file is created and deleted between scans. The trigger does not occur because the file does not exist when the directory is scanned.

Example: Monitor the Creation of a File

This example monitors for the creation of a file named monthly.log. The job completes when the file is created.

```
insert_job: ft_job
job_type: FT
machine: ftagt
watch_file: "c:\data\monthly.log"
watch_file_type: CREATE
```

Example: Monitor the Creation and Stability of a File

This example monitors for the creation of a file named monthly.log. The job completes when the file is created and has not changed for consecutive watch_no_change polling intervals.

```
insert_job: ft_job
job_type: FT
machine: ftagt
watch_file: "c:\data\monthly.log"
watch_file_type: GENERATE
watch_no_change: 2
```

Example: Monitor for the Deletion of Files with Names Beginning with Pay

This example continuously monitors the /usr/data/ directory for files that have names beginning with pay. When all files that have a name beginning with pay are deleted, the job completes successfully.

```
insert_job: ftjob
job_type: FT
machine: ftagt
watch_file: /usr/data/pay*
watch_file_type: DELETE
continuous: Y
```

Example: Monitor for an Increase in File Size by a Specific Amount

In this example, the unixagt agent monitors the record file in the credit directory. If the record file expands in size by 200,000 bytes or more, the job completes.

```
insert_job: ft_unix_delta
job_type: FT
machine: unixagt
watch_file: /credit/record
watch_file_type: EXPAND
watch_file_change_type: DELTA
watch_file_change_value: 200000
```

Example: Check that a File Does Not Exist in a Directory

In this example, the unixagt agent checks for the file named vacation in directory /start/term/. If the vacation file does not exist in that directory, the job completes. If the vacation file exists in that directory, the job fails.

```
insert_job: ft_unix_notexist
job_type: FT
machine: unixagt
watch_file: /start/term/vacation
watch_file_type: NOTEXIST
```

Example: Monitor for a Decrease in File Size to a Specific Amount

In this example, the unixagt agent monitors the distribute file in the /cash/items directory. If the file size shrinks to 1000 bytes or smaller, the job completes.

```
insert_job: ft_unix_shrink
job_type: FT
machine: unixagt
watch_file: /cash/items/distribute
watch_file_type: SHRINK
watch_file_change_type: SIZE
watch_file_change_value: 1000
```

Example: Monitor for Updates to Any of the Files in a Directory and its Subdirectories

This example monitors all the files in the /usr/data/ directory and its subdirectories. When any file is updated in any of the monitored directories, the job completes.

```
insert_job: ft_unix_update
job_type: FT
machine: unixagt
watch_file: "/usr/data/*"
watch_file_type: UPDATE
watch_file_recursive: Y
```

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following File Trigger job attributes have default values:

continuous

Specifies whether the job monitors the file continuously for a specified condition.

Default: N (The job immediately checks for the condition and completes.)

watch_file_recursive

Specifies whether the job monitors for file activity in the specified directory only or in the specified directory and all of its subdirectories.

Default: N (The job does not monitor subdirectories.)

watch_file_change_type

Specifies the type of change to detect when monitoring a file for an increase or decrease in size..

Default: SIZE (The job monitors the file to reach a specified size.)

watch_file_type

Specifies the type of file activity to monitor for.

Default: CREATE (The job monitors for the creation of a file.)

watch_no_change

Defines the number of minutes the file must remain unchanged to satisfy the monitor condition.

Default: 1 (The file must remain unchanged for 1 minute.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Override Default Values

The watch_file_type and watch_no_change attributes in the following job definition override the default values.

In this example, the unixagt agent monitors the analysis file in the /research directory. If the file size expands to 1 byte or more and remains unchanged for 120 minutes or more, the job completes.

```
insert_job: ft_unix_nochange
job_type: FT
machine: unixagt
watch_file: /research/analysis
watch_file_type: EXPAND
watch_file_change_type: SIZE
watch_file_change_value: 1
watch_no_change: 120
```

Monitor a File Continuously

You can define a File Trigger job to monitor a file continuously for a CREATE, GENERATE, EXPAND, UPDATE, DELETE, or SHRINK condition. Each time the condition occurs, an alert is written to the scheduler log file (event_demon.$AUTOSERV on UNIX and event_demon.%AUTOSERV% on Windows).

If a job monitors for the deletion of files, the job completes when all the monitored files are deleted or it is completed manually. For all other conditions, the job continues to monitor until it is completed manually.

Note: You cannot define File Trigger jobs to monitor a file continuously for the EXIST or NOTEXIST conditions.

To monitor a file continuously

1. Define a File Trigger job (see page 266).
2. Add the following attribute to the job definition:

   continuous

   Specifies whether the job monitors the file continuously for a specified condition.
3. Run the job.

   The specified file is monitored continuously.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Continuously Monitor the Same File Using Multiple File Trigger Jobs

In this example, two File Trigger jobs monitor the same file for a change in size by 10 KB. One job monitors for an increased change in size and the other job monitors for a decreased change in size. The jobs are independent and do not relate to each other in any way.

```
insert_job: ftjob1
job_type: FT
machine: ftagent
watch_file: "c:\data\totals"
watch_file_type: EXPAND
continuous: Y
watch_file_change_type: DELTA
watch_file_change_value: 10240 /* The value must be entered in bytes (10 x 1024 bytes
= 10240) */

insert_job: ftjob2
job_type: FT
machine: ftagent
watch_file: "c:\data\totals"
watch_file_type: SHRINK
continuous: Y
watch_file_change_type: DELTA
watch_file_change_value: 10240 /* The value must be entered in bytes (10 x 1024 bytes
= 10240) */
```

Suppose that the initial file size of totals (c:\data\totals) is 100 KB and it changes as follows:

100 KB (initial size), 80 KB, 90 KB, 110 KB, 50 KB, 60 KB

The following triggers occur:

- Because the file trigger type is EXPAND, the first job (ftjob1) writes an alert to the scheduler log file once when the file size changes from 90 KB to 110 KB.
- Because the file trigger type is SHRINK, the second job (ftjob2) writes an alert to the scheduler log file twice. The triggers occur when the file shrinks from 100 KB to 80 KB and then again when the file shrinks from 110 KB to 50 KB.

The jobs end when they are forced to complete.

Monitor a File that is Owned by a UNIX Owner or Group

You can define a File Trigger job to monitor a file that is owned by a specific UNIX owner or group. If the file is not owned by the specified owner or group, the following occurs:

- The job continues monitoring if the file trigger type is CREATE or GENERATE.
- The job completes if the file trigger type is DELETE or NOTEXIST.
- The job fails if the file trigger type is EXPAND, EXIST, SHRINK, or UPDATE.

Note: You can monitor a file owned by a specific owner or group on i5/OS if the file is not a QSYS object.

To monitor a file that is owned by a UNIX owner or group

1. Define a File Trigger job (see page 266).
2. Add one or more of the following attributes to the job definition:

   watch_file_groupname

   Specifies the name of the group that owns the file to be monitored.

   watch_file_owner

   Specifies the user ID that owns the file to be monitored.
3. Run the job.

   The specified file is monitored.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor for the Creation of a File that Is Owned by a Specified UNIX User ID

This example monitors for the creation of a file named payroll owned by JDOE.

```
insert_job: ft_job
job_type: FT
machine: ftagt
watch_file: /usr/data/payroll
watch_file_type: CREATE
watch_file_owner: JDOE
```

The job completes as follows:

- If the file does not exist when the job starts, the job does not complete until the file is created.
- If the file exists and the owner of the file is JDOE when the job is readied, the job completes immediately.
- If the file exists or is created, but the owner of the file is not JDOE, the job does not complete. It waits until all specified criteria are satisfied, including the owner criteria. If the owner of the file changes to JDOE, the job completes.

Example: Monitor for the Existence of a File that Is Owned by a Specified UNIX Group

Suppose that you want a job to monitor for the existence of the /data/payroll.dat file that is owned by the UNIX group ACCTS:

- If the file exists and the file is owned by the group ACCTS when the job is readied, the job completes immediately.
- If the file exists and the file is not owned by the group ACCTS when the job is readied, the job fails.

```
insert_job: ft_unixgroup
job_type: FT
machine: unixagent
watch_file: /data/payroll.dat
watch_file_type: EXIST
watch_file_groupname: ACCTS
```

Configure the Agent to Run File Trigger Jobs as an External Process

File Trigger jobs typically run as threads under the CA Workload Automation Agent process. If you want to monitor files with names that contain variables, or if you want to monitor a file on a remote computer, you must run the job as an external process.

To configure the agent to run File Trigger jobs as an external process, ask your agent administrator to set the following parameters in the agent's agentparm.txt file:

```
filemonplugin.runexternal=true
oscomponent.default.user=user
oscomponent.default.password=password
```

filemonplugin.runexternal=true

Runs File Trigger jobs run as an external process.

oscomponent.default.user=*user*

(Optional) Specifies the default operating system user ID that all jobs on the agent computer run under. This user ID must have access to all files monitored by all File Trigger jobs.

Note: If this parameter is not set, File Trigger jobs run under the user ID that started the agent.

oscomponent.default.password=*password*

(Optional) Specifies the password for the default user ID.

Resolve File Names That Contain Variables

You can define a File Trigger job that monitor a file whose name contains variables. To resolve the variables in the file name, the agent runs the File Trigger job as an external process.

To resolve file names that contain variables

1. Configure the agent to run File Trigger jobs as an external process (see page 276).
2. Define a File Trigger job (see page 266).
3. Add the following attribute:

   watch_file

   Specifies the path to and name of one or more files to monitor. You can specify file names that contain variables.

   Example: $AUTOSYS/reports/august2011.out

4. Run the job.

   The specified files are monitored.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor for the Existence of a File

This example monitors for the existence of the august2011.out file on the unixagt machine. The job monitors the directory specified by the $MY_PATH variable. The job runs under the user that started the unixagt agent. The filemonplugin.runexternal agent parameter is set to true. If the file exists in the directory, the job completes. If the file does not exist in that directory, the job fails.

```
insert_job: ft_job
job_type: FT
machine: unixagt
watch_file: $MY_PATH/reports/august2011.out
watch_file_type: EXIST
```

Monitor a File on a Remote UNIX Computer

You can define a File Trigger job to monitor remote files across a UNIX network using a specified user ID.

To monitor a file on a remote UNIX computer

1. Configure the agent to run File Trigger jobs as an external process (see page 276).
2. Define a File Trigger job (see page 266).
3. Add the following attribute:

   watch_file

   Specifies the path to and name of one or more files to monitor. You can specify file names that contain variables and remote files on a UNIX network.

   Example: $AUTOSYS/reports/august2011.out
4. Run the job.

   The specified files are monitored.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor for the Existence of a File

This example monitors for the existence of the august2011.out file on the unixagt machine. The job monitors the directory specified by the $MY_PATH variable. The job runs under the user that started the unixagt agent. The filemonplugin.runexternal agent parameter is set to true. If the file exists in the directory, the job completes. If the file does not exist in that directory, the job fails.

```
insert_job: ft_job
job_type: FT
machine: unixagt
watch_file: $MY_PATH/reports/august2011.out
watch_file_type: EXIST
owner: jsmith
```

Monitor a File on a Remote Windows Computer

You can define a File Trigger job to monitor a file on a remote Windows system if the agent runs as a Windows service under the local system account.

To monitor a file on a remote Windows computer

1. Verify with your agent administrator that the agent is running as a Windows service under the local system account.
2. (Optional) Ask your agent administrator to set the following parameters in the agent's agentparm.txt file:

   ```
   oscomponent.default.user=user
   oscomponent.default.password=password
   ```

   oscomponent.default.user=*user*

   Specifies the default operating system user ID that all jobs on the agent computer run under. This user ID must have access to the monitored files.

   Notes:

   - The watch_file_win_user attribute overrides this parameter.
   - If this parameter is not set, you must specify the watch_file_win_user attribute in the File Trigger job definition.

   oscomponent.default.password=*password*

   Specifies the password for the default user ID.

   Note: If this parameter is not set, you must specify the watch_file_win_user attribute in the File Trigger job definition and define the corresponding user ID and password via autosys_secure.

3. Ask your CA Workload Automation AE administrator to define a user ID and password on CA Workload Automation AE that has access to the file on the remote Windows computer.
4. Define a File Trigger job (see page 266).
5. Add the following attributes to the job definition:

   watch_file

   Specifies the path to and name of one or more files to monitor. Specify a UNC (Universal Naming Convention) name. A UNC name is the name of a file or other resource that begins with two backslashes (\\), indicating that it exists on a remote computer.

   watch_file_win_user

   Specifies the user ID and the domain the user ID belongs to. This user ID must have access to the remote files in the UNC path. This attribute overrides the oscomponent.default.user parameter on the agent.

6. Run the job.

   The specified file on the remote Windows computer is monitored.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor for an Update to a File on a Remote Windows Computer

This example monitors for an update to the payroll.dat file on a remote Windows computer named CYBNT. The job runs under JSMITH, which is a user ID on CYBNT and is in the CYBDOM domain. JSMITH is defined on the scheduling manager and has access to the AccountingFiles directory. The winagent machine must have been defined to CA Workload Automation AE with the **opsys: windows** attribute.

```
insert_job: ft_remotefile
job_type: FT
machine: winagent
watch_file: \\CYBNT\AccountingFiles\payroll.dat
watch_file_type: UPDATE
watch_file_win_user: CYBDOM\JSMITH@CYBDOM
```

Chapter 10: File Watcher Jobs

This section contains the following topics:

File Watcher Jobs

A File Watcher job is similar to a Command job. However, instead of starting a user-specified command on a client computer, a File Watcher job starts a process that monitors for the existence and size of a specific operating system file. When that file reaches the specified minimum size and is no longer growing in size, the File Watcher job completes successfully, indicating that the file has arrived.

Using File Watcher jobs provides a means of integrating events that are external to CA Workload Automation AE into the processing conditions of jobs. For example, assume a file must be downloaded from a mainframe, and it is expected to arrive after 2:00 a.m. After it arrives, a batch job is run to process it, possibly even starting a whole sequence of jobs.

You could set up a File Watcher job to start at 2:00 a.m., wait for the arrival of the specified file, and exit. You could also set up the batch job so that the completion of the File Watcher job is its only starting condition.

Note: To run these jobs, your system requires one of the following:

- CA WA Agent for UNIX, Linux, Windows, or i5/OS
- Legacy agent for Unicenter AutoSys Job Management r4.5.1 through r11

Define a File Watcher Job

You can define a File Watcher job to monitor for the existence and size of a file. CA Workload Automation AE considers the watched file complete when the file reaches the minimum file size specified in the watch_file_min_size attribute and the file reaches a “steady state” during the polling interval. A steady state indicates that the watched file has not grown during the specified interval.

Note: To run these jobs, your system requires one of the following:

- CA WA Agent for UNIX, Linux, Windows, or i5/OS
- Legacy agent for Unicenter AutoSys Job Management r4.5.1 through r11

To define a File Watcher job

1. Insert a job and specify the following attributes in the definition:

   job_type: FW

   Specifies that the job type is File Watcher.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   watch_file

   Specifies the path to and name of one or more files to monitor.

2. (Optional) Specify optional File Watcher attributes:
   - job_class
   - job_terminator
   - watch_file_min_size

     Note: The default size to monitor is 0 (zero). If you do not specify the watch_file_min_size attribute, the job completes if the file exists. You can specify this attribute to override the default setting.
   - watch_interval

     Notes:
     - On the legacy agent, if you do not specify the watch_interval attribute, the job checks the file every 60 seconds (the default). You can specify the watch_interval attribute to override the default setting.
     - On the CA Workload Automation Agent, if you do not specify the watch_interval attribute, the job checks the file every 30 seconds (or the time specified in the agent's filemonplugin.sleepperiod parameter). If you specify this attribute for a FW job that is submitted to an the CA WA agent, the agent uses the value as a “no-change” or steady interval. The steady interval means that once the file condition is satisfied the file must remain steady for the duration specified by the watch_interval attribute.

3. (Optional) Specify common attributes that apply to all job types.

   The File Watcher job is defined.

Notes:

- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor a File Every 120 Seconds

This example monitors the watch_file.log file on the winagent computer. The job completes when the file reaches 10,000 bytes and maintains a steady state for at least 120 seconds (30 seconds for the agent’s global poll interval plus 90 seconds for the watch_interval).

```
insert_job: fw_job
job_type: FW
machine: winagent
watch_file: “c:\tmp\watch_file.log”
watch_file_min_size: 10000
watch_interval: 90
```

Example: Monitor a File Every 60 Seconds on a Legacy Agent

This example monitors the watch_file.log file on the unixagent computer. The unixagent is a legacy agent. The job completes when the file reaches 10000 bytes and maintains a steady state for 60 seconds.

```
insert_job: fw_job
job_type: FW
machine: unixagent
watch_file: /tmp/watch_file.log
watch_file_min_size: 10000
watch_interval: 60
```

More information:

Insert a Job (see page 87)

Chapter 11: FTP Jobs

This section contains the following topics:

FTP Jobs

Using your agent, you can automate File Transfer Protocol (FTP) transfers with an FTP job. The FTP job can upload data to or download data from an existing FTP server or another agent running as an FTP server. The FTP job always acts as an FTP client.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

You can use an FTP job to automate the following:

- Download ASCII, binary, or EBCDIC (i5/OS only) files from a remote FTP server to your agent computer.
- Upload ASCII, binary, or EBCDIC (i5/OS only) files from your agent computer to a remote FTP server.

Your agent administrator can set up the agent to run as an FTP client, FTP server, or both.

EBCDIC File Transfers

The EBCDIC transfer type applies to CA WA Agent for i5/OS only.

For the QSYS file system on i5/OS systems, you can only transfer members of FILE objects.

Note: For more information about FTP restrictions on i5/OS systems, see the IBM documentation.

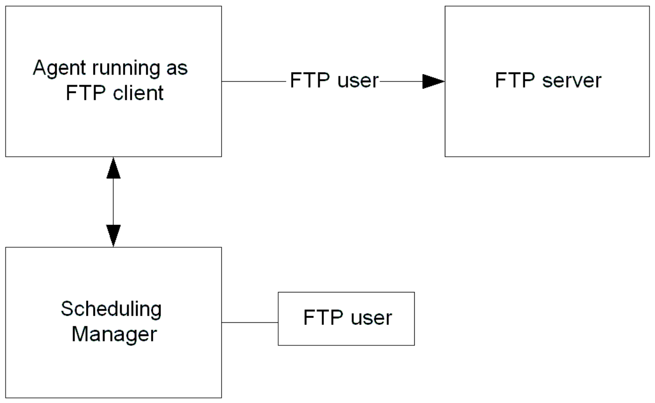
Wildcard Characters in File Names

You can use wildcards in file names for ASCII, binary, and EBCDIC transfers. The asterisk (*) is a wildcard for zero or more characters and the question mark (?) is a wildcard for a single character.

Running the Agent as an FTP Client

If the agent runs as an FTP client, the agent can log in to remote FTP servers and download files from and upload files to those servers.

The following diagram shows the relationship between an agent running as an FTP client, the scheduling manager, and an FTP server:



Note: The FTP user ID used to connect to the FTP server must be defined on the scheduling manager.

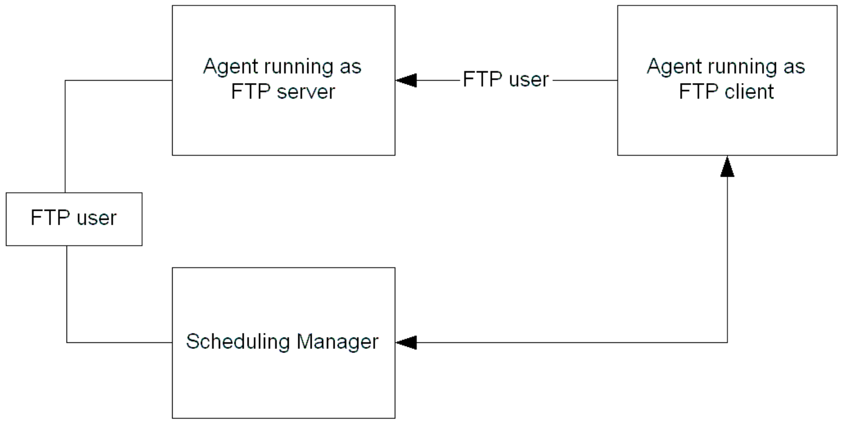
When the agent runs as an FTP client only, other FTP clients (such as other agents) cannot log in to the agent to transfer files. To let other FTP clients log in and transfer files, the agent administrator needs to set up the agent to run as an FTP server.

Running the Agent as an FTP Server

The agent supports a built-in FTP server capability. The agent administrator can enable the agent to act as a generic FTP server in addition to its other roles. This server adheres to the security rules established for the agent.

If the agent runs as an FTP server, clients can log in to the agent and transfer files.

The following diagram shows the relationship between an agent running as an FTP server, the scheduling manager, and another agent running as an FTP client:



Note: The FTP user ID used to connect to the agent running as an FTP server must be defined on that agent and the scheduling manager.

If the agent is configured as an FTP server, the agent can handle ASCII, binary, and EBCDIC file transfers, wildcard requests, simple GET and PUT requests for single files, and MGET and MPUT requests for multiple files.

The agent has a secure store of FTP server user IDs and associated passwords. The ftpusers.txt file, located in the directory where the agent is installed, stores these user IDs and their corresponding hashed passwords.

The agent running as an FTP server does not support anonymous FTP requests. For audit purposes, the agent provides a detailed log of all FTP requests.

Note: For more information about enabling the agent to act as a generic FTP server, contact your agent administrator and see the *CA Workload Automation Agent for UNIX, Linux, or Windows Implementation Guide*.

FTP User IDs and Passwords

All FTP user IDs and passwords must be defined on CA Workload Automation AE by using the autosys_secure command. When you define an FTP job, you must specify an FTP user ID using the owner attribute, or use the default owner value. This user ID is used to connect to the FTP server for the file transfer.

If an agent runs as an FTP server, the FTP user ID and password must also be defined on that agent.

Note: For more information about the autosys_secure command, see the *Reference Guide*. For more information about defining user IDs on the agent, see the *CA Workload Automation Agent for UNIX, Linux, or Windows Implementation Guide*.

Define an FTP Job

You can define an FTP job to automate FTP transfers. The output is directed to the spool file through an FTP server.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

To define an FTP job

1. Insert a job and specify the following attributes in the definition:

   job_type: FTP

      Specifies that the job type is FTP.

   machine

      Specifies the name of an existing machine definition where the agent runs.

   ftp_local_name

      Specifies the destination of the file (if downloading) or the source location of the file (if uploading).

   ftp_remote_name

      Specifies the source location of the file (if downloading) or the destination of the file (if uploading).

   ftp_server_name

      Specifies the DNS name or IP address of a remote server.

2. (Optional) Specify optional FTP attributes:
   - ftp_command
   - ftp_compression
   - ftp_local_user
   - ftp_server_port
   - ftp_transfer_direction
   - ftp_transfer_type
   - ftp_use_SSL
   - job_class
3. (Optional) Specify the following attribute:

   owner

   Specifies the user ID with the authority to download the file from the remote FTP server or upload the file to the remote FTP server.

   Note: The job runs under the owner of the job. The password associated with the owner must be defined via autosys_secure.
4. (Optional) Specify common attributes that apply to all job types.

   The FTP job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Download a File from a UNIX Computer to a Windows Computer

The FTP job in this example downloads a file from a UNIX server with IP address 172.16.0.0 and port 21 to a Windows computer.

```
insert_job: FTPT1A
job_type: FTP
machine: ftpagent
ftp_server_name: 172.16.0.0
ftp_server_port: 21
ftp_transfer_direction: DOWNLOAD
ftp_transfer_type: A
ftp_remote_name: /u1/ftpdata/textfile
ftp_local_name: "c:\ftpfiles"
owner: ftpuser@172.16.0.0
```

Example: Download a Binary File on UNIX

The FTP job in this example uses a binary transfer.

```
insert_job: FTPBINARY
job_type: FTP
machine: unixagent
ftp_server_name: hpunix
ftp_server_port: 5222
ftp_transfer_direction: DOWNLOAD
ftp_transfer_type: B
ftp_remote_name: /u1/qatest/ftpdata/binaryfile
ftp_local_name: /export/home/qatest/ftpdata/transf.bin
owner: test@hpunix
```

Example: Download a QSYS EBCDIC-encoded File

This example downloads an EBCDIC-encoded file named DATAFILE in the QSYS file system from an i5/OS system to another i5/OS system. The file names are specified in the path format.

```
insert_job: EBCDIC_FILE
job_type: FTP
machine: I5AGENT
ftp_server_name: i5agent
ftp_server_port: 5222
ftp_transfer_direction: DOWNLOAD
ftp_transfer_type: E
ftp_remote_name: /QSYS.LIB/DATALIB.LIB/DATAFILE.FILE/DATA.MBR
ftp_local_name: /QSYS.LIB/ESPLIB.LIB/DOWNLOAD.FILE/DATA.MBR
owner: test@i5agent
```

More information:

Insert a Job (see page 87)

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following FTP job attributes have default values:

ftp_compression

Specifies the data compression level (0-9).

Default:

- 0 (The data is not compressed.)
- ftp.data.compression agent parameter. This parameter overrides the default 0 setting.

ftp_transfer_direction

Specifies the file transfer direction.

Default: DOWNLOAD (The job transfers files from the remote server to the agent computer.)

ftp_transfer_type

Specifies the type of data involved in an FTP transfer (B for binary, A for ASCII, or E for EBCDIC).

Default: B (The job performs a binary data transfer.)

ftp_use_ssl

Specifies whether to transfer the data with Secure Sockets Layer (SSL) communication or regular communication.

Default:

- FALSE (The job uses regular communication.)
- ftp.client.ssl agent parameter. This parameter overrides the default FALSE setting.

owner

Specifies the FTP user ID that the job runs under.

Default: The user defining the job (The FTP transfer runs under the owner of the job.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Perform an ASCII Data Transfer

The ftp_transfer_type and owner attributes in the following job definition override the default values. The job transfers an ASCII file under the testuser account.

The FTP job in this example downloads an ASCII file named textfile from the remote UNIX server to the /export/home/ftpfiles/ftpdata/textfile_dn directory on the agent computer.

```
insert_job: DOWNLOAD_SINGLE
job_type: FTP
machine: RAGENT
ftp_server_name: hprsupp
ftp_server_port: 5222
ftp_transfer_direction: DOWNLOAD
ftp_transfer_type: A
ftp_remote_name: /u1/files/ftpdata/textfile
ftp_local_name: /export/home/ftpfiles/ftpdata/textfile_dn/textfile
owner: testuser@hprsupp
```

Transfer Files Using SSL FTP

If the agent FTP server supports SSL FTP and the agent FTP client has SSL FTP configured, you can securely transfer data using Secure Sockets Layer (SSL) communication.

To transfer files using SSL FTP

1. Define an FTP job (see page 288).
2. Add the following attribute to the job definition:

   ftp_use_ssl

   Specifies whether to transfer the data with Secure Sockets Layer (SSL) communication or regular communication.
3. Run the job.

   The files are transferred using the specified communication setting.

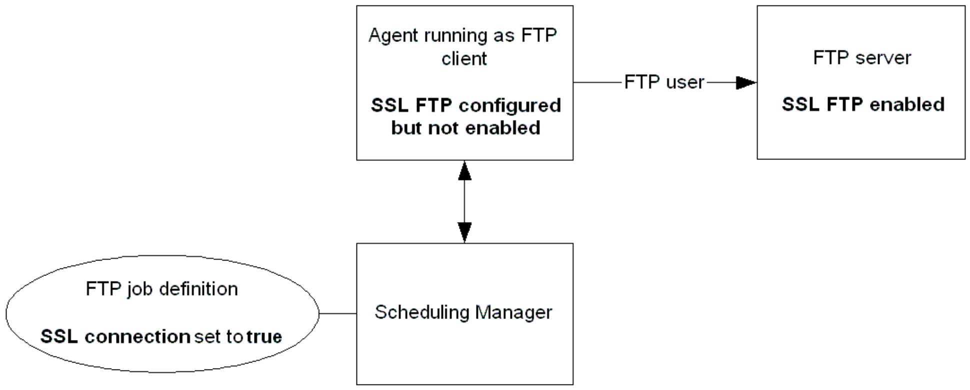
Notes:

- To transfer data using SSL, the FTP server must have SSL FTP enabled and the FTP client must have SSL configured (SSL FTP can be enabled or disabled).
- If you do not specify the ftp_use_ssl attribute, the data is transferred using the default FTP setting (regular FTP or SSL FTP) set on the agent FTP client as follows:
  - If the agent FTP client has SSL FTP enabled, all FTP jobs on that agent automatically use SSL FTP.

    Note: If the FTP server does not support SSL FTP, you must set the ftp_use_ssl attribute to FALSE. Otherwise, the job will fail.
  - If the agent FTP client has SSL FTP disabled, all FTP jobs on that agent automatically use regular FTP.

Example: Upload a File from a Local Computer to a Remote Windows Server Using SSL FTP

In this example, the agent runs on a local computer as an FTP client and has SSL FTP configured but not enabled. The remote Windows server has SSL FTP configured and enabled.

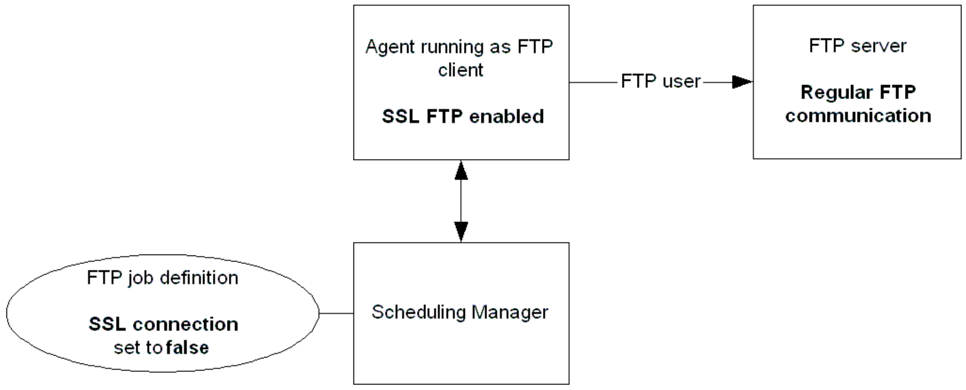

Agent running as FTP client
SSL FTP configured but not enabled
FTP user
FTP server
SSL FTP enabled
FTP job definition
SSL connection set to true
Scheduling Manager

The following FTP job uploads the filename.txt file from the agent FTP client to the c:\uploaded_files directory on a remote Windows server. The ftp_use_SSL attribute is set to TRUE to transfer the file securely. Although the agent FTP client does not have SSL FTP enabled, the file will be uploaded using SSL FTP because the configuration requirements are met (the agent FTP client has SSL FTP configured and the FTP server has SSL FTP enabled).

```
insert_job:FTP_UPLOAD
job_type: FTP
machine: winagent
owner: user1@winserver
ftp_server_name: winserver
ftp_server_port: 21
ftp_transfer_direction: UPLOAD
ftp_use_SSL: TRUE
ftp_remote_name: "c:\uploaded_files\filename.txt"
ftp_local_name: "d:\files_to_upload\filename.txt"
```

Example: Download a File from a Remote UNIX Server That Does Not Support SSL FTP to a Local Computer That Supports SSL FTP

In this example, the agent runs on a local computer as an FTP client and has SSL FTP enabled (all FTP jobs on the agent computer run using SSL FTP by default). The remote UNIX server does not support SSL FTP.


Agent running as FTP client
SSL FTP enabled
FTP user
FTP server
Regular FTP communication
FTP job definition
SSL connection set to false
Scheduling Manager

The following FTP job downloads the filename.txt file from the remote UNIX server to the d:\downloaded_files directory on the local computer. Because the FTP server does not support SSL FTP, the ftp_use_SSL attribute is set to FALSE so the job does not fail.

```
insert_job: FTP_DOWNLOAD
job_type: FTP
machine: winagent
owner: user1@hpunix
ftp_server_name: hpunix
ftp_server_port: 5222
ftp_transfer_direction: DOWNLOAD
ftp_use_SSL: FALSE
ftp_remote_name: /files_to_download/filename.txt
ftp_local_name: "d:\downloaded_files\filename.txt"
```

Compress Data for FTP

When running FTP workload between an FTP client and FTP server that are both run on the agent software, you can compress the data for the transfer by specifying the compression level in the job definition.

Note: If the compression level is specified and the FTP server or the FTP client does not run on the agent, the data will be transferred without compression.

To compress the data that you want to transfer

1. Define an FTP job (see page 288).
2. Add the following attribute to the job definition:

   ftp_compression

   Specifies the data compression level. The compression level is a one-digit value from 0 to 9, where 0 is no data compression and 9 is the best data compression.

   Note: The effectiveness of the compression is dependent upon the data. Compressing the data may not result in faster transfer times. The overhead of compressing and uncompressing the data may exceed the time saved from sending smaller amount of data.
3. Run the job.

   The files are transferred using SSL FTP.

Example: Compress a File

The local computer in this example has an agent running as an FTP client. The remote server has an agent running as an FTP server. Both computers operate on a low bandwidth network.

The following FTP job downloads a large file named largefile.txt from the remote server to the FTP client. The computers are on a low bandwidth network, so the data is compressed at compression level 3.

```
insert_job: FTPJOB
job_type: FTP
machine: ftpagent
ftp_server_name: aixunix
ftp_server_port: 5222
ftp_transfer_direction: DOWNLOAD
ftp_compression: 3
ftp_remote_name: /files_to_download/largefile.txt
ftp_local_name: "c:\downloaded_files\largefile.txt"
owner: ftpuser@aixunix
```

Send Site-Specific FTP Commands to FTP Servers

When you define an FTP job, you can specify one or more commands to execute prior to file transfer. You can use this feature to send site-specific FTP commands to FTP servers.

To send a site-specific FTP command to an FTP server

1. Define an FTP job (see page 288).
2. Add the following attribute to the job definition:

   ftp_command

   Defines a command that is to be executed prior to file transfer.

   Note: This value includes commands such as locsite but does not include commands such as cd or lcd.

3. (Optional) Specify additional ftp_command attributes for each command that you want to run.
4. Run the job.

   The site-specific FTP command is sent to the FTP server.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Send FTP Commands to an FTP Server

This example sends two FTP commands to the FTP server prior to transferring a file.

```
insert_job: CYBJK.FTP
job_type: FTP
machine: ftpagent
ftp_server_name: ftp.ca.com
ftp_server_port: 21
ftp_transfer_direction: DOWNLOAD
ftp_transfer_type: B
ftp_compression: 8
ftp_remote_name: /pub/cazip.exe
ftp_local_name: /tmp/bla
ftp_command: locsite blksize=FB
ftp_command: locsite automount
owner: user1@ftp.ca.com
```

Verify the FTP Job Status

You can verify that the transfer completed successfully without file corruption.

To verify that the transfer completed successfully, check the job's spool file for the following responses:

- If the data was transferred using SSL FTP, the spool file contains the following response:

  ```
  AUTH TLS
  234 AUTH command OK. Initializing SSL connection.
  ```

- If the data was compressed and transferred without file corruption, the spool file contains a response as follows:

  ```
  Downloaded 81920/26119 bytes (original/compressed) in 0.161 seconds, 496.89
  Kbytes/sec.
  ```

- If the file was downloaded successfully, the spool file contains the following response:

  ```
  Download successful
  ```

Chapter 12: i5/OS Jobs

This section contains the following topics:

i5/OS Jobs

The i5/OS job lets you run a program or issue a command on an i5/OS system. You can run i5/OS jobs in the following file systems:

- Root file system
- Open systems file system (QOpenSys)
- Library file system (QSYS)

Note: To run these jobs, your system requires CA WA Agent for i5/OS.

You can specify the following details in an i5/OS job definition:

- Library name, library list, and current library for running a program
- The i5/OS job name, options under which the job will run, where it will run, and which user will run it
- Ending exit value of the program, such as a severity code

You can define parameter values that you want to pass to a program at the time the program is invoked.

Note: Default values may be set for certain parameters, such as the i5/OS user ID that the jobs run under. Contact your agent administrator about the parameters set in the agentparm.txt file.

Running UNIX Workload on a System i5 Computer

In addition to scheduling native i5/OS jobs, you can schedule most UNIX workload, such as UNIX scripts, in the PASE environment on i5/OS.

To run both native and UNIX jobs on the same i5/OS computer, you must install two i5/OS agents and configure the oscomponent.targetenvironment parameter in the agentparm.txt file to handle the appropriate job type. For more information about configuring the parameter, see the *CA Workload Automation AE UNIX Implementation Guide* or *Windows Implementation Guide*.

Note: For more information about UNIX workload that can run in the PASE environment, see the IBM i5/OS documentation.

i5/OS Naming Conventions

When specifying i5/OS paths and names in your workload, you can use the following naming conventions, depending on where the file is located on the i5/OS system:

- Root file system

  To specify a file in the root file system, use UNIX path and file formats.

- Open systems file system (QOpenSys)

  To specify a file in QOpenSys, use UNIX path and file formats. QOpenSys file names are case-sensitive.

- Library file system (QSYS)

  To specify an object in QSYS, use one of the following formats (unless described differently in the job definition syntax):

  – Path format

    /QSYS.LIB/*library*.LIB/*object*.*type*/

    To specify *FILE objects, use the following format:

    /QSYS.LIB/*library*.LIB/*object*.FILE/*member*.MBR

  – i5/OS standard format

    library/*object*/*type*

    To specify *FILE objects, use the following format:

    library/*object*/*FILE(*member*)

    Note: *FILE is optional when *member* is specified. That is, you can specify a file member using the following format:

    library/*object*(*member*)

Notes:

- The values for *library*, *object*, *type*, and *member* can be up to 10 characters each.
- You can use *ALL to match any name.
- You can use *FIRST for *member*.
- You can use generic names for *library* and *object*.

Define an i5/OS Job

You can define an i5/OS job to schedule workload to run on an i5/OS system. The job can run a program or an i5/OS command. You can run i5/OS jobs in the root file system, open systems file system (QOpenSys), and library file system (QSYS).

Note: To run these jobs, your system requires CA WA Agent for i5/OS.

To define an i5/OS job

1. Insert a job and specify the following attributes in the definition:

   job_type: I5

   Specifies that the job type is i5/OS.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   i5_name

   Specifies the program, the source file for the program, or the command that you want to run.

   Note: The value must correspond to the i5_action value. If you do not specify the i5_action attribute, the job interprets the corresponding i5_name value as a command by default.

2. (Optional) Specify optional i5/OS attributes:

   - fail_codes
   - i5_action
   - i5_cc_exit
   - i5_curr_lib
   - i5_job_desc

- i5_job_name
- i5_job_queue
- i5_lda
- i5_lib
- i5_library_list
- i5_others
- i5_params
- i5_process_priority
- job_class
- max_exit_success
- success_codes

3. (Optional) Specify common attributes that apply to all jobs.

   The i5/OS job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Run an i5/OS Command

This example runs the command named DSPJOBLOG on the i5agent computer.

```
insert_job: i5job_runcmd
job_type: I5
machine: i5agent
i5_name: DSPJOBLOG
```

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following i5/OS job attributes have default values:

i5_action

Specifies whether to run a program or issue a command.

Default: COMMAND (The job interprets the i5_name value to be a command.)

i5_cc_exit

Specifies the type of exit code returned by an i5/OS job.

Default: *SEVERITY (The job sends the ending severity code as the exit code.)

i5_job_desc

Specifies the job description for the submitted program.

Default: os400.default.jobdname agent parameter, if specified

i5_job_queue

Specifies the i5/OS job queue for the submitted program.

Note: os400.default.jobqname agent parameter, if specified

i5_process_priority

Specifies the process priority of the i5/OS job.

Default: NORMAL

owner

Specifies the user ID that the job runs under.

Default: The user ID who invokes jil to define the job (the owner of the job)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Override Default Values

This example runs an i5/OS program. A default job queue is defined in the agent's agentparm.txt file. The i5_action and i5_job_queue attributes in this job definition override the default values.

```
insert_job: i5job_lib
job_type: I5
machine: i5agent
i5_action: RUN_PROGRAM
i5_name: PAYLOAD
i5_job_queue: /QYS.LIB/QBASE.LIB/JQUEUE.JOBQ
```

Pass Positional Parameters

When running workload, you might need to pass data between jobs and across platforms. You can pass positional parameters to an i5/OS program in your job definition. Positional parameters are variables that can be passed to a program at the time the program is invoked. The parameters are assigned in the order they are passed.

To pass positional parameters to an i5/OS program

1. Define an i5/OS job (see page 301).
2. Add the following attribute to the job definition:

   i5_params

   Defines the parameter values that you want to pass to the program at the time the program is invoked.

3. Run the job.

   The specified positional parameters are passed to the i5/OS program.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Pass Multiple Parameters to an i5/OS Job

This example passes six parameters to an i5/OS program named PAYJOB. The parameter VALUE C is enclosed with double quotation marks because it contains a space.

```
insert_job: i5job_lib
job_type: I5
machine: i5agent
i5_action: RUN_PROGRAM
i5_name: PAYJOB
i5_params: ABC 1 P "VALUE C" X r
```

Use a User's Library List

The agent uses the library list in the job's job description by default. However, if the user is defined, you can set up your job definition to use the user's library list instead.

To use a user's library list

1. Define an i5/OS job (see page 301).
2. Do *one* of the following:
   - Add the following attribute to the job definition:

     i5_curr_lib: (*USRPRF)

     Specifies that the job uses the user's current library when it runs.
   - Add the following attributes to the job definition:

     i5_job_desc: (*JOBD)

     Specifies that the job uses the job description assigned to the user to access the library list.

     i5_library_list: (*USRPRF)

     Specifies that the job accesses the user’s library list when it runs.
3. Run the job.

   The job uses the user's library list.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Pass Keyword Parameters to SBMJOB

When CA Workload Automation AE submits a job to the i5/OS system, the job must pass through the SBMJOB command to execute. The following JIL attributes map to parameters for the SBMJOB command:

JIL Attributes	SBMJOB Parameters
i5_user	USER
i5_job_desc	JOBD
i5_library_list	INLLIBL
i5_job_queue	JOBQ
i5_curr_lib	CURLIB
i5_job_name	JOB

You can also pass additional keyword parameters, such as OUTQ(*JOBD), to the SBMJOB command.

To pass an SBMJOB command keyword and value combination

1. Define an i5/OS job (see page 301).
2. Add the following attribute to the job definition:

   i5_others

   Specifies SBMJOB command keyword and value combinations.
3. Run the job.

   The specified keywords and values are passed to the SBMJOB command.

Notes:

- The special values for these SBMJOB parameters, such as *USRPRF and *JOBD, also apply to the JIL attributes. You can use these special values in your job definitions. For more information about the SBMJOB parameters and their special values, see the IBM documentation.
- For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Specify the Printer and Output Queue for an i5/OS Job

This example runs a program named PAYJOB on an i5/OS system. The printer and output queue information is taken from the job definition.

```
insert_job: i5job_lib
job_type: I5
machine: i5agent
i5_action: RUN_PROGRAM
i5_name: PAYJOB
i5_others: PRTDEV=*JOBD,OUTQ=*JOBD
```

Responding to Suspended Jobs that Require Manual Intervention

A program run on an i5/OS system can temporarily suspend itself until it receives additional feedback from the end user. In this scenario, CA WA Agent for i5/OS notifies the scheduler that a manual response is required. The scheduler sets the job status to the WAIT_REPLY state. A WAIT_REPLY_ALARM is raised with text containing the query of the i5/OS program as well as the set of expected responses. For the i5/OS job to resume program execution, you must send a response to the job. For example, suppose that you schedule an i5/OS job to save data in a file. If the file already contains data, the i5/OS program prompts you to confirm that the data in the file can be overwritten.

To respond to suspended jobs that require manual intervention, issue the following command:

```
sendevent -J job_name -E REPLY_RESPONSE -r response
```

The response is sent to the CA WA Agent for i5/OS and the job resumes running.

Returning a Job's Exit Status to CA Workload Automation AE

A job's exit code indicates whether the job completed successfully or failed. By default, the agent sends the job's ending severity code to CA Workload Automation AE when a job completes. CA Workload Automation AE interprets an exit code of zero (0) as job success and any other number as job failure.

In addition to sending the job's ending severity code, you can return a job's exit status in other ways. For example, you can send the return code of an ILE program or module as the exit status or you can specify a user-defined exit code of 100 as success.

You can return a job's exit status to CA Workload Automation AE using the following methods:

- Send a program's return code using the i5_cc_exit attribute
- Send a user-defined exit code using the success_codes or fail_codes attribute
- Return an exit status using the EXIT_SUCCESS and EXIT_FAILURE macros

Send a Program's Return Code

When a job completes, the agent sends the job's exit code to CA Workload Automation AE. By default, the agent sends the job's ending severity code as the job's exit code.

Instead of sending the job's ending severity code, the agent can send the return code of one of the following:

- An ILE program or module
- An OPM program

For example, if your job runs an ILE C, ILE RPG, OPM RPG, or OPM Cobol program that contains an exit or return statement, the agent can send that return code as the exit code.

To send a program's return code

1. Define an i5/OS job (see page 301).
2. Add *one* of the following attributes to the job definition:

   i5_cc_exit: *USER

   Specifies that the return code of an ILE program or module is sent as the exit code.

   i5_cc_exit: *PROGRAM

   Specifies that the return code of an OPM program is sent as the exit code.
3. Run the job.

   The program's return code is sent instead of the job's ending severity code.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Send an OPM COBOL Program's Return Code as the Job's Exit Code

This example runs an OPM COBOL program named PAYROLL. The agent sends the PAYROLL program's return code to CA Workload Automation AE.

```
insert_job: i5job_returnOPM
job_type: I5
machine: i5agent
i5_action: RUN_PROGRAM
i5_name: PAYROLL
i5_cc_exit: *PROGRAM
```

Example: Send an ILE C Program's Return Code as the Job's Exit Code

This example runs a C language program named SALARY. The agent sends the SALARY program’s return code to CA Workload Automation AE. Ending severity codes of 1 or 10 indicate job success.

```
insert_job: i5job_returnC
job_type: I5
machine: i5agent
i5_action: RUN_PROGRAM
i5_name: SALARY
i5_cc_exit: *USER
success_codes: 1,10
```

Note: The i5 system always writes the job’s ending severity code to the job’s spool file. You can check the spool file for the job’s ending severity code for more information about the job status. For example, suppose that the C program's return code indicates failure, but the ending severity code shown in the spool file is 10, which might indicate that the job ran with a minor issue. Assuming that this issue can be ignored, you can indicate ending severity codes of 10 as job success using the success_codes attribute.

Send a User-defined Exit Code

By default, CA Workload Automation AE interprets an exit code of 0 (zero) as job success and any other number as job failure. However, you can map exit codes other than 0 as job success.

To send a user-defined exit code

1. Define an i5/OS job (see page 301).
2. Add *one* of the following attributes to the job definition:

   success_codes

   Defines which exit codes indicate job success.

   Default: 0 (zero)

   fail_codes

   Defines which exit codes indicate job failure.

   Default: Any non-zero exit code
3. Run the job.

   The specified user-defined exit code is sent.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Send Exit Code 100 as Success

This example runs the PAYPROG program. The program is considered to have completed successfully if it returns an exit code of 1 or 100.

```
insert_job: i5job_succ
job_type: I5
machine: i5agent
i5_action: RUN_PROGRAM
i5_name: PAYPROG
i5_cc_exit: *PROGRAM
success_codes: 1,100
```

Example: Send Exit Code 40 as Failure

This example runs the RECPROG program. The program is considered to have failed if if it returns an exit code of 40. All other exit codes in the range from 50 to 255 indicate job success.

```
insert_job: i5job_fail
job_type: I5
machine: i5agent
i5_action: RUN_PROGRAM
i5_name: RECPROG
i5_cc_exit: *PROGRAM
fail_codes: 40
success_codes: 50-255
```

Specify Data for a Local Data Area

The local data area is a temporary 1024-byte storage area that exists for the duration of an i5/OS job. You can use the local data area to pass data to the job and to other programs that run as part of the job. When the job is submitted, the agent initializes the local data area with the specified data. When the job completes, the local data area is destroyed automatically by the operating system.

To specify data for a local data area

1. Define an i5/OS job (see page 301).
2. Add the following attribute to the job definition:

   i5_lda

   Specifies data for the local data area in an i5/OS job.
3. Run the job.

   The data is specified for a local data area.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Specify Data for the Local Data Area in Hexadecimal Format

This example defines an i5/OS job with data for the local data area. When the job is submitted, the agent initializes the local data area with the hexadecimal data abcd. When the job completes, the local data area is destroyed automatically by the operating system.

```
insert_job: i5job_lda
job_type: I5
machine: i5agent
i5_action: COMMAND
i5_name: IVP
i5_lda: x'abcd'
```

Chapter 13: Monitoring Jobs

This section contains the following topics:

Monitoring Jobs

Monitoring jobs let you monitor different aspects of your system.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

You can define the following monitoring jobs:

CPU Monitoring

Lets you monitor CPU usage.

Disk Monitoring

Lets you monitor disk space.

IP Monitoring

Lets you monitor an IP address.

Process Monitoring

Lets you monitor process execution.

Text File Reading and Monitoring

Lets you search a text file for a string.

Windows Event Log Monitoring

Lets you monitor a Windows event log.

Windows Service Monitoring

Lets you monitor the status of Windows services.

Define a CPU Monitoring Job

You can define a CPU Monitoring (OMCPU) job to monitor the CPU usage of the computer the specified agent is installed on. By default, the job monitors for available CPU and completes when the specified conditions are met.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

To define a CPU Monitoring job

1. Insert a job and specify the following attributes in the definition:

   job_type: OMCPU

   Specifies that the job type is CPU Monitoring.

   machine

   Specifies the name of an existing machine definition where the agent runs.

2. (Optional) Specify optional CPU Monitoring attributes:
   - cpu_usage
   - inside_range
   - job_class
   - job_terminator
   - monitor_mode
   - no_change
   - poll_interval

3. Specify *at least one* of the following attributes if monitor_mode is set to WAIT (the default) or CONTINUOUS:
   - lower_boundary
   - upper_boundary

4. (Optional) Specify common attributes that apply to all job types.

   The CPU Monitoring job is defined. When the job runs, it monitors for available CPU on the specified machine and completes when the CPU usage value falls within the lower and upper boundaries.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Check Available CPU Immediately

This example monitors available CPU usage on the unixagent computer. The job checks the CPU usage immediately and reports the value in percent.

```
insert_job: omcpu_job
job_type: OMCPU
machine: unixagent
monitor_mode: NOW
```

Example: Monitor When the Available CPU Reaches 50 Percent

This example monitors the CPU available on the unixagent computer. The default monitor mode is WAIT, so a lower boundary, upper boundary, or both boundaries must be specified. In this example, the lower boundary is specified and the default upper boundary (100 percent) is used. When the available CPU usage is within 50 and 100 percent, the job completes.

```
insert_job: omcpu_job
job_type: OMCPU
machine: unixagent
lower_boundary: 50
```

CPU Monitoring Modes

When you define a CPU monitoring job, you can define the mode it runs in. Depending on the mode, CPU monitoring jobs can do the following:

- Monitor CPU usage and complete when the specified conditions are met (WAIT mode). This is the default.
- Record the CPU usage at the time the job runs. In this case, the job runs only once (NOW mode).
- Monitor CPU usage and trigger an alert if the CPU usage meets the defined criteria (CONTINUOUS mode). For example, a job can trigger an alert when the computer is using between 80 and 100 percent of CPU, and the job continues to run until you manually terminate it.

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following CPU Monitoring job attributes have default values:

cpu_usage

Specifies whether the job monitors the available or used CPU processing capacity.

Default: FREE (The job monitors for available CPU.)

inside_range

Specifies whether the job completes (or triggers if monitoring continuously) when the value of CPU usage is inside or outside the specified boundaries.

Default: TRUE (The job completes or triggers if the value of the CPU usage is within the lower and upper boundaries.)

lower_boundary

Defines the minimum amount of CPU usage to monitor for in percent.

Default: 0 (The job monitors the CPU usage between zero and the upper boundary.)

monitor_mode

Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

Default: WAIT (The job waits until the specified conditions are met before completing.)

poll_interval

Defines the interval (in seconds) between successive scans of the CPU usage.

Default: objmon.scaninterval agent parameter (This parameter is automatically set to 10. The job polls the CPU usage every 10 seconds during continuous monitoring.)

upper_boundary

Defines the maximum amount of CPU usage to monitor for in percent.

Default: 100 (The job monitors the CPU usage between the lower boundary and 100 percent.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor CPU Usage Until Condition Successful

This example checks the CPU usage and completes successfully when the used CPU is within 80 and 95 percent. The cpu_usage, monitor_mode, lower_boundary, and upper_boundary attributes in this job definition override the default values.

```
insert_job: omcpu_job
job_type: OMCPU
machine: winagent
cpu_usage: USED
inside_range: TRUE
monitor_mode: WAIT
lower_boundary: 80
upper_boundary: 95
```

Examples: Monitoring CPU Usage

The following examples are CPU Monitoring Jobs:

Example: Monitor Used CPU

This example monitors the used CPU on the unixagent computer. The job completes when the used CPU is less than 20 percent or greater than 80 percent.

```
insert_job: omcpu_job
job_type: OMCPU
machine: unixagent
lower_boundary: 20
upper_boundary: 80
inside_range: FALSE
cpu_usage: USED
```

Example: Monitor CPU Availability Within a Range

This example continuously monitors the CPU available on the unixagent computer. The job polls the CPU usage every 60 seconds. Each time the available CPU is within 75 and 95 percent, an alert is written to the scheduler log file. The job continues monitoring the CPU usage until it is ended manually.

```
insert_job: omcpu_job
job_type: OMCPU
machine: unixagent
cpu_usage: FREE
inside_range: TRUE
poll_interval: 60
monitor_mode: CONTINUOUS
lower_boundary: 75
upper_boundary: 95
```

Example: Monitor Available CPU Continuously

This example continuously monitors the CPU available on the winagt computer. An alert is written to the scheduler log file each time the available CPU is less than 25 percent or greater than 75 percent, and the CPU usage changes by more than 10 percent. If the change in value is less than or equal to 10, the job does not register a change.

```
insert_job: omcpu_job
job_type: OMCPU
machine: winagt
cpu_usage: FREE
lower_boundary: 25
upper_boundary: 75
inside_range: FALSE
no_change: 10
monitor_mode: CONTINUOUS
```

The following table shows when alerts would be triggered with and without the no_change value:

Time	CPU	Is the Alert Triggered When No Change is not specified?	Is the Alert Triggered When No Change is specified at 10 percent?
14:00:01	25 percent	No. Available CPU must be below 25 percent or above 75 percent.	No. Available CPU must be below 25 percent or above 75 percent.
14:00:02	20 percent	Yes. Available CPU is below 25 percent.	Yes. Available CPU is below 25%.
14:00:03	19 percent	Yes. Available CPU is below 25 percent.	No. Available CPU remains below 25 percent, but the change from the last reading is only 1 percent.
14:00:04	8 percent	Yes. Available CPU is below 25 percent.	Yes. CPU usage has changed 12 percent from the last time the alert was triggered.
14:00:05	19 percent	Yes. Available CPU is below 25 percent.	Yes. CPU usage has changed 11 percent from the last time the alert was triggered.

Time	CPU	Is the Alert Triggered When No Change is not specified?	Is the Alert Triggered When No Change is specified at 10 percent?
14:00:06	32 percent	No. Available CPU must be below 25 percent or above 75 percent.	No. Although CPU usage changed by more than 10 percent, it no longer falls within the range defined by the lower_boundary and upper_boundary fields. It is not below 25 percent or above 75 percent.

Define a Disk Monitoring Job

On UNIX and Windows systems, you can define a Disk Monitoring (OMD) job to monitor the available or used space on a disk or logical partition. On i5/OS systems, you can define a Disk Monitoring job to monitor storage space in the file systems mounted on the i5/OS operating system.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

To define a Disk Monitoring job

1. Insert a job and specify the following attributes in the definition:

   job_type: OMD

   Specifies that the job type is Disk Monitoring.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   disk_drive

   Specifies the path to the disk, logical partition, or auxiliary storage pool to be monitored.

2. (Optional) Specify optional Disk Monitoring attributes:
   - disk_format
   - disk_space
   - inside_range
   - job_class
   - job_terminator
   - monitor_mode
   - no_change
   - poll_interval

3. Specify *one or both* of the following attributes if monitor_mode is set to WAIT (the default) or CONTINUOUS:
   - lower_boundary
   - upper_boundary

4. (Optional) Specify common attributes that apply to all job types.

   The Disk Monitoring job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Check Available Disk Space Immediately

This example monitors available disk space on a local UNIX partition. The job checks the disk usage immediately and reports the value in megabytes.

```
insert_job: unix_freemb
job_type: OMD
machine: unixagent
disk_drive: /export/home
disk_format: MB
monitor_mode: NOW
```

Example: Monitor When the Available Disk Space Reaches 50 Percent

This example monitors the disk space available in /export/home on the unixagent computer. The default monitor mode is WAIT, so a lower boundary, upper boundary, or both boundaries must be specified. In this example, the lower boundary is specified and the default upper boundary (100 percent) is used. When the available disk space is within 50 and 100 percent, the job completes.

```
insert_job: omd_job
job_type: OMD
machine: unixagent
disk_drive: /export/home
lower_boundary: 50
```

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following Disk Monitoring job attributes have default values:

disk_format

Specifies the unit of measurement used to monitor available or used disk space.

Default: PERCENT (The job monitors disk usage by percent.)

disk_space

Specifies whether the job monitors for available or used disk space.

Default: FREE (The job monitors for available disk space.)

inside_range

Specifies whether the job completes (or triggers if monitoring continuously) when the value of disk usage is within or outside the specified boundaries.

Default: TRUE (The job completes or triggers if the value of the disk usage is within the lower and upper boundaries.)

monitor_mode

Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

Default: WAIT (The job waits until the specified conditions are met before completing.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor for Used Disk Space Outside of a Range

The inside_range and disk_space attributes in the following job definition override the default values.

This example monitors the C drive on a Windows computer for used space. When the value of disk space used falls below 16 percent or exceeds 95 percent, the job completes.

```
insert_job: omd_win
job_type: OMD
machine: winagent
disk_drive: C
disk_format: PERCENT
lower_boundary: 16
upper_boundary: 95
inside_range: FALSE
disk_space: USED
```

Examples: Monitoring Disk Space

The following examples are Disk Monitoring Jobs:

Example: Monitor Available Space on a UNIX Partition

This example monitors available disk space on a local UNIX partition. The job checks the disk usage immediately and reports the value in megabytes.

```
insert_job: unix_freemb
job_type: OMD
machine: unixagent
disk_drive: /export/home
disk_format: MB
monitor_mode: NOW
```

Example: Continuously Monitor Used Space on a UNIX Partition

This example continuously monitors used disk space on a local UNIX partition. Each time the used disk space falls between 90 and 100 percent, an alert is written to the scheduler log file. The job continues monitoring the disk space until it is ended manually.

```
insert_job: unix_used
job_type: OMD
machine: unixagent
disk_drive: /export/home
disk_space: USED
disk_format: PERCENT
monitor_mode: CONTINUOUS
lower_boundary: 90
```

Example: Monitor Used Space on a Windows Drive

This example monitors the used disk space on a local Windows C drive. When the used disk space falls below 16 percent or exceeds 95 percent, the job completes.

```
insert_job: win_used
job_type: OMD
machine: winagent
disk_drive: C
disk_format: PERCENT
disk_space: USED
monitor_mode: WAIT
lower_boundary: 16
upper_boundary: 95
inside_range: FALSE
```

Example: Monitor the System Auxiliary Storage Pool on an i5/OS Computer

This example continuously monitors the system auxiliary storage pool on an i5/OS computer. Each time the used disk space falls between 90 and 100 percent, an alert is written to the scheduler log file. The job continues monitoring the disk space until it is ended manually.

```
insert_job: i5_used
job_type: OMD
machine: i5agent
disk_drive: /
disk_space: USED
disk_format: PERCENT
monitor_mode: CONTINUOUS
lower_boundary: 90
upper_boundary: 100
```

Example: Monitor Available Disk Space That Changes by At Least 100 KB

This example continuously monitors the available disk space in kilobytes (KB) on the local Windows C drive. When the available space is in the 35000000 to 36000000 KB range, the first alert is written to the scheduler log file.

Subsequently, an alert is triggered each time the available disk space is within the specified boundaries and the disk usage changes by more than 100 KB. If the amount of change is less than or equal to 100 KB, the job does not register a change.

```
insert_job: omd_job
job_type: OMD
machine: winagent
disk_drive: C
disk_space: FREE
disk_format: KB
lower_boundary: 35000000
upper_boundary: 36000000
inside_range: TRUE
no_change: 100
monitor_mode: CONTINUOUS
```

The following table shows four sequential scans:

Scan	Scanned Amount (Kilobytes)	Does the Trigger Occur?
1	35018896	Yes.
2	35018900	No. Comparing scan 2 to scan 1, the delta value is only 4 KB. This scanned amount will not be included in the next calculation.
3	35018795	Yes. Comparing scan 3 to scan 1, the delta value is greater than 100 KB. The delta value of the next scan will be calculated using the scan 3 value of 35018795.
4	36000001	No. The scanned amount is outside the lower and upper boundary range.

Define an IP Monitoring Job

You can define an IP Monitoring (OMIP) job to monitor an IP address or a port at an IP address.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

To define an IP Monitoring job

1. Insert a job and specify the following attributes in the definition:

   job_type: OMIP

   Specifies that the job type is IP Monitoring.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   ip_host

   Specifies the DNS name or IP address.

2. (Optional) Specify optional IP Monitoring attributes:
   - ip_port
   - ip_status
   - job_class
   - job_terminator
   - monitor_mode

3. (Optional) Specify common attributes that apply to all job types.

   The IP Monitoring job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor a Device Until it Stops

This example monitors the device APPARCL. When the device stops running, the job completes.

```
insert_job: omip_stop
job_type: OMIP
machine: SYSAG
ip_host: APPARCL
monitor_mode: WAIT
```

Monitoring Remote IP Addresses on UNIX

To monitor remote IP addresses through the agent, the agent must run as root (on the CA WA Agent for UNIX, Linux, or Windows) or under a profile with sufficient authority to use the system ping command (on the CA WA Agent for i5/OS). If the agent runs as a user without root privileges, a job that monitors a remote IP address shows complete but with the following message in the status field and transmitter.log:

```
Ping (ip address) insufficient privilege
```

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following IP Monitoring job attributes have default values:

ip_status

Specifies the status of the IP address to monitor.

Default: STOPPED (The job monitors the IP address for a stopped status.)

monitor_mode

Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

Default: NOW (The job checks for the conditions immediately and completes.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor for a RUNNING Status Immediately

The ip_status attribute in the following job definition overrides the default value.

This example monitors a device at IP address 172.31.255.255 and port 7510. When the job runs, it immediately checks if the device is running. If the device is running, the job completes successfully.

```
insert_job: omip_job
job_type: OMIP
machine: monagt
ip_host: 172.31.255.255
ip_port: 7510
monitor_mode: NOW
ip_status: RUNNING
```

Examples: Monitoring an IP Address

The following examples are IP Monitoring Jobs:

Example: Monitor an IP Address for a Stopped Status

This example monitors a device with DNS name myhost. When the device stops running, the job completes.

```
insert_job: omip_job
job_type: OMIP
machine: monagt
ip_host: myhost
ip_status: STOPPED
monitor_mode: WAIT
```

Example: Monitor an Agent IP Address Specified in Dotted Decimal Format

This example checks whether an application at a specific IP address using a specific port is running. The IP address is 172.24.2.20 and the input port is 9401. When the job runs, it checks the status immediately and completes if the application is running.

```
insert_job: omip_job
job_type: OMIP
machine: SYSAG
ip_host: 172.24.2.20
ip_port: 9401
ip_status: RUNNING
monitor_mode: NOW
```

Example: Monitor a Device at a Specific IP Address and Port Number

This example checks whether the localhost device at port number 5800 is running. When the job runs, it checks the status immediately and completes successfully if the device is running.

```
insert_job: omip_running
job_type: OMIP
machine: SYSAG
ip_host: localhost
ip_port: 5800
ip_status: RUNNING
monitor_mode: NOW
```

Define a Process Monitoring Job

You can define a Process Monitoring (OMP) job to monitor the status of a process on the computer where the agent is installed.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

To define a Process Monitoring job

1. Insert a job and specify the following attributes in the definition:

   job_type: OMP

   Specifies that the job type is Process Monitoring.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   process_name

   Specifies the name of the process to be monitored.

2. (Optional) Specify optional Process Monitoring attributes:
   - job_class
   - job_terminator
   - monitor_mode
   - process_status
3. (Optional) Specify common attributes that apply to all job types.

   The Process Monitoring job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor a Running Process

This example monitors the nlnotes process. By default, the job checks if the process is stopped. If the process is stopped, the job completes successfully. If the process is running, the job continues monitoring it until it stops.

```
insert_job: omp_unix
job_type: OMP
machine: unixagt
process_name: nlnotes
monitor_mode: WAIT
```

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following Process Monitoring job attributes have default values:

monitor_mode

Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

Default: NOW (The job checks for the conditions immediately and completes.)

process_status

Specifies the status of the process to be monitored.

Default: STOPPED (The job checks if the process is stopped.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor for a STOPPED Status

The monitor_mode attribute in the following job definition overrides the default value.

This example monitors the cybAgent.exe process. The job checks the process status immediately and completes successfully if the process is stopped. If the process is running, the job continues monitoring until the process stops.

```
insert_job: omp_win
job_type: OMP
machine: winagt
process_name: "c:\Program files\Agent\cybAgent.exe"
process_status: STOPPED
monitor_mode: WAIT
```

Examples: Monitoring Processes

The following examples are Process Monitoring Jobs:

Example: Monitor Multiple Instances

This example monitors the Microsoft SQL Server processes of two instances using the full path name. When the server process stops, the job monitoring that instance completes successfully. The first job monitors the sqlserver.exe process in the …\MSSQL.1\MSSQL\Binn directory. The second job monitors the sqlserver.exe process in the …\MSSQL.2\MSSQL\Binn directory.

```
insert_job: mon_sql_server_instance1
job_type: OMP
machine: mssqlserver
process_name: "C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Binn\sqlservr.exe"
process_status: stopped
monitor_mode: wait
```

```
insert_job: mon_sql_server_instance2
job_type: OMP
machine: mssqlserver
process_name: "C:\Program Files\Microsoft SQL
Server\MSSQL.2\MSSQL\Binn\sqlservr.exe"
process_status: stopped
monitor_mode: wait
```

Example: Monitor the Agent Process on i5/OS

This example monitors the CYBAGENT process on an i5/OS computer. The job checks the process status immediately and completes successfully if the process is running.

```
insert_job: omp_i5_onejob
job_type: OMP
machine: i5agt
process_name: 123456/CYBESPU/CYBAGENT
process_status: RUNNING
monitor_mode: NOW
```

Example: Monitor Multiple i5/OS Processes That Have Similar Names

This example monitors all processes running on an i5/OS computer under the JDOE user profile and whose names start with CALC. When all of these processes stop running, the job completes successfully.

```
insert_job: omp_i5
job_type: OMP
machine: i5agt
process_name: *ALL/JDOE/CALC*
process_status: STOPPED
monitor_mode: WAIT
```

Define a Text File Reading and Monitoring Job

You can define a Text File Reading and Monitoring (OMTF) job to search a text file on a Windows, UNIX, or i5/OS computer for a text string. For example, you can monitor a log file for an error message after a script executes.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

To define a Text File Reading and Monitoring job

1. Insert a job and specify the following attributes in the definition:

   job_type: OMTF

   Specifies that the job type is Text File Reading and Monitoring.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   text_file_filter

   Defines the text string to search for. You can specify the text string as a regular expression.

text_file_name

Specifies the path to and name of the text file to search.

2. (Optional) Specify optional Text File Reading and Monitoring attributes:
   - encoding
   - job_class
   - job_terminator
   - lower_boundary
   - monitor_mode
   - text_file_filter_exists
   - text_file_mode
   - time_format
   - time_position
   - upper_boundary
3. (Optional) Specify common attributes that apply to all job types.

   The Text File Reading and Monitoring job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor a File for a Specified String

This example monitors the entire transactions.log file for the ERROR MESSAGE string. The job waits for the string to be found and then completes successfully.

```
insert_job: textfile_job
job_type: OMTF
machine: monagt
text_file_name: /export/home/logs/transactions.log
text_file_filter: ERROR MESSAGE
```

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following Text File Reading and Monitoring job attributes have default values:

encoding

Specifies the name of the character set used to encode the data in the file.

Default: US-ASCII (The job monitors the file as US-ASCII.)

monitor_mode

Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

Default: WAIT (The job waits until the specified conditions are met before completing.)

text_file_mode

Specifies the search mode when monitoring a text file.

Default: LINE (The job searches for the text in the specified line boundaries.)

text_file_filter_exists

Specifies whether the job monitors the text file to check if the text string exists or does not exist.

Default: TRUE (The job checks if the text string exists.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Search for Text Between Regular Expressions

The text_file_mode attribute in the following job definition overrides the default LINE monitor mode.

This example searches the /export/home/systemagent/agentparm.txt file. The search starts at the first line that contains the word "agent" at the beginning of the line (as specified by \A in the regular expression specified for lower_boundary) and until it finds the string "level=2" at the end of a line (as specified by \Z in the regular expression specified for upper_boundary).

```
insert_job: omtf_unix_line
job_type: OMTF
machine: monagt
text_file_name: /export/home/systemagent/agentparm.txt
text_file_filter: \.0/MAIN$
text_file_mode: REGEX
lower_boundary: \Aagent
upper_boundary: level=2\Z
monitor_mode: NOW
```

Example: Specify a Data Encoding Value

The encoding attribute in the following job definition overrides the default US-ASCII data encoding value.

This example monitors a text file that contains data encoded in the ISO Latin Alphabet No. 1 (also named ISO-LATIN-1). The job checks the text file immediately and completes successfully if the specified string is found. If the string is not found, the job fails.

```
insert_job: textfile_job
job_type: OMTF
machine: monagt
text_file_name: /export/home/logs/transactions.log
text_file_filter: ERROR MESSAGE
text_file_mode: LINE
lower_boundary: 1
upper_boundary: 50
encoding: ISO-8859-1
monitor_mode: NOW
```

Examples: Monitoring a Text File for Specified Text

The following examples are Text File Reading and Monitoring Jobs:

Example: Search for a Regular Expression

In this example, the text string contains regular expression pattern matching syntax. The search range is also a regular expression as indicated by the text_file_mode attribute.

```
insert_job: textfile_job3
job_type: OMTF
machine: monagt
text_file_name: /export/home/agentdir/agentparm.txt
text_file_filter: ^\w{4,10}\.
text_file_mode: REGEX
lower_boundary: \A\W\sE
```

The regular expression can be interpreted as follows:

- ^ or \A — match only at the beginning of string (line)
- \Z or $ — match only at the end of string
- \w — a word character [a-zA-Z0-9]
- \W — a non-word character
- \s — a whitespace character
- {4,10} — match at least 4 times but not more than 10 times

  To illustrate the last item (4, 10), consider the syntax:

  ```
  text_file_filter: b1{1,3}c
  ```

  Evaluating this expression yields the following conditions:

  - The line contains the text b1.
  - Numeric 1 should exist at least once, but not more than three times.
  - The specified text string must be followed by the letter c.

Example: Search for a String in a File Starting at a Specified Line

This example searches the c:\ca\log file in line mode. The job starts searching the content from line 143 of the file. The upper boundary is not defined, so the job searches to the last line of the file. The job completes successfully if the ERROR MESSAGE string is found.

```
insert_job: omtf_line
job_type: OMTF
machine: monagt
text_file_name: "c:\ca\log"
text_file_filter: ERROR MESSAGE
text_file_mode: LINE
lower_boundary: 143
monitor_mode: NOW
```

Example: Search for a String in a File When the Search Mode is REGEX

This example searches the c:\ca\log file in regular expression mode. The lower boundary is not defined, so the job searches the content from the first line of the file to the upper boundary (a line that contains the word service). The job completes successfully if the ARCHIVE string is found.

```
insert_job: omtf_regex
job_type: OMTF
machine: monagt
text_file_name: "c:\ca\log"
text_file_filter: ARCHIVE
text_file_mode: REGEX
upper_boundary: service
monitor_mode: NOW
```

Example: Search for a String in a File When the Search Mode is DATETIME

This example searches the /export/home/logs/transmitter.log file in date and time mode. The job searches the content between May 20, 2010 at midnight and May 27, 2010 at 11:59 p.m. The date and time values are defined using the format specified in the time_format attribute. The job completes successfully if the transmitted string is found.

```
insert_job: omtf_timedate
job_type: OMTF
machine: monagt
text_file_name: /export/home/logs/transmitter.log
text_file_filter: transmitted
text_file_mode: DATETIME
lower_boundary: "Thu May 20 00:00:00.000 EDT 2010"
upper_boundary: "Thu May 27 23:59:59.999 EDT 2010"
time_format: "EEE MMM dd HH:mm:ss.SSS zzz yyyy"
time_position: 12
monitor_mode: NOW
```

Example: Monitor a Text File Continuously

This example searches the transmitter.log file for the text string "Warning".

```
insert_job: textfile_job
job_type: OMTF
machine: monagt
text_file_name: /export/home/log/transmitter.log
text_file_filter: Warning
text_file_mode: LINE
lower_boundary: 25
text_file_filter_exists: TRUE
monitor_mode: CONTINUOUS
```

When the job first runs, it searches the content between line 25 and the end of the file. An alert is written to the scheduler log file the first time that the string is found. In other words, suppose that the file contains multiple occurrences of "Warning" between lines 25 to the end of the file, as follows:

```
.
.
.
25
26 Warning
27
28 Warning
29
30
31 Warning
.
.
.
EOF
```

When the job first runs, the trigger only occurs at the first occurrence of the text (line 26). Subsequently, the job continues monitoring only the new data that is *appended* to the file. An alert is triggered each time the string is found in the appended data.

Note: Alerts are not triggered for new occurrences of the "Warning" string in the data that has already been searched. For example, suppose that the job has already searched lines 25 to 100 of the file. The file is then modified to include "Warning" on line 30. During continuous monitoring, an alert is *not* triggered for that occurrence.

This job runs until it is completed manually.

Example: Search for a Text String on an i5/OS Computer

This example searches for a text string in the DATA member of a QSYS file object on an i5/OS computer. The job searches the content between lines 1 and 20. The job completes successfully if the string is found.

```
insert_job: textfile_job3
job_type: OMTF
machine: monagt
text_file_name: /QSYS.LIB/LIBRARY.LIB/RESULTS.FILE/DATA.MBR
text_file_filter: Create file failed
text_file_mode: LINE
lower_boundary: 1
upper_boundary: 20
monitor_mode: NOW
```

Example: Check If a String Does Not Exist in a File

This example searches lines 1 to 200 of the transmitter.log file for the text string "Warning". If the string is *not* found, the job completes successfully. If the string is found, the job fails.

```
insert_job: textfile_job
job_type: OMTF
machine: monagt
text_file_name: /export/home/log/transmitter.log
text_file_filter: Warning
text_file_mode: LINE
lower_boundary: 1
upper_boundary: 200
text_file_filter_exists: FALSE
monitor_mode: NOW
```

Define a Windows Event Log Monitoring Job

You can define a Windows Event Log Monitoring (OMEL) job to monitor a Windows event log on the computer where the agent is running. The monitor returns the most recent event available or continuously monitors for events in a particular Windows event log.

Note: To run these jobs, your system requires CA WA Agent for Windows.

To define a Windows Event Log Monitoring job

1. Insert a job and specify the following attributes in the definition:

   job_type: OMEL

   Specifies that the job type is Windows Event Log Monitoring.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   win_log_name

   Specifies the name of the event log.

2. (Optional) Specify optional Windows Event Log Monitoring attributes:
   - job_class
   - job_terminator
   - monitor_mode
   - win_event_category
   - win_event_computer
   - win_event_datetime
   - win_event_description
   - win_event_id
   - win_event_op
   - win_event_source
   - win_event_type
3. (Optional) Specify common attributes that apply to all job types.

   The Windows Event Log Monitoring job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor an Application Event Log

This example monitors an Application event log. By default, the job waits until an Error event occurs before it completes.

```
insert_job: eventlog_job
job_type: OMEL
machine: monagt
win_log_name: Application
```

Types of Event Logs

The Windows Event Log Monitoring job only monitors event logs maintained by the operating system and available in the Event Viewer. Windows operating systems record events in at least three types of logs, including the following:

Application log

The application log contains events logged by applications or programs. For example, a database program might record a file error in the application log.

System log

The system log contains events logged by the Windows system components. For example, the failure of a driver or other system component to load during startup is recorded in the system log.

Security log

The security log can record security events (such as valid and invalid logon attempts) and events related to resource use (such as creating, opening, or deleting files).

For more information on Windows logs, select Start, Settings, Control Panel, Administrative Tools, Event Viewer. Select any of the three log categories and double-click to view its property page.

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following Windows Event Log Monitoring job attributes have default values:

monitor_mode

Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

Default: WAIT (The job waits until the specified conditions are met before completing.)

win_event_type

Specifies the event type to monitor in the Windows event log.

Default: ERROR (The job monitors for the Error event type.)

win_event_op

Specifies a comparison operator against the value of a Windows Event ID.

Default: EQ (The job monitors for an Event ID that is equal to the specified value.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor an Event Log Continuously for Info Events

The monitor_mode, win_event_op, and win_event_type attributes in the following job definition override the default values.

This example monitors the event log for applications continuously for all instances of an INFO event type, where the event source is LLDSAPNT223, the event description contains the word started, and the event ID is less than or equal to 4000. Each time the specified conditions occur, an alert is written to the scheduler log file.

```
insert_job: eventlog_job
job_type: OMEL
machine: monagt
win_log_name: Application
win_event_source: LLDSAPNT223
win_event_category: None
win_event_type: INFO
win_event_op: LE
win_event_id: 4000
win_event_description: started
monitor_mode: CONTINUOUS
```

Examples: Monitoring a Windows Event Log

The following examples are Windows Event Log Monitoring Jobs:

Example: Monitor an Application Log That Occurs on or after a Specified Date

This example monitors an application log that occurs any time on or after January 12, 2010, 6:30 a.m. When the job finds an application log that occurs any time on or after that date and time, the job completes successfully.

```
insert_job: win_eventlog
job_type: OMEL
machine: winagent
win_log_name: Application
win_event_type: info
win_event_category: None
win_event_source: LLDSAPNT223
win_event_datetime: "20100112 06:30:00"
```

Example: Monitor Events with IDs Equal to 0

This example checks for an event ID number less than 1. The job returns the first application event from the application log that has an event ID equal to 0.

```
insert_job: eventlog_job
job_type: OMEL
machine: monagt
win_log_name: Application
win_event_op: LT
win_event_id: 1
```

Example: Monitor a System Event Log

This example monitors a system event log for an event type of WARN, event source of MrxSmb, and event category of None.

```
insert_job: eventlog_job
job_type: OMEL
machine: monagt
win_log_name: System
win_event_type: WARN
win_event_source: MrxSmb
win_event_category: None
```

Example: Monitor a Security Event Log for Audit Success Events

In this example, the security log is monitored for a successful audit of a security access attempt. The event category is System Event, the term succeeded is excluded, but the words Audit and log are included in the event description.

```
insert_job: eventlog_job
job_type: OMEL
machine: monagt
win_log_name: Security
win_event_type: AUDITS
win_event_category: System Event
win_event_source: Service Control Manager
win_event_description: "-succeeded +Audit log"
```

Example: Monitor a System Event Log for Particular Errors

In this example, the event description must include the words conflict and state as indicated by the plus signs but must exclude the words deny, master, or browser as indicated by the minus sign. The plus sign is the default and is optional.

```
insert_job: eventlog_job1
job_type: OMEL
machine: monagt
win_log_name: System
win_event_type: ERROR
win_event_description: "+conflict +state -deny -master -browser"
```

Example: Monitor an Application Log for Events Indicating Normal Shutdown

In this example, the event description must include the words Normal shutdown. This example does not use the plus sign, and by default, the specified words are included in the search.

```
insert_job: eventlog_job2
job_type: OMEL
machine: monagt
win_log_name: Application
win_event_type: INFO
win_event_description: "Normal shutdown"
```

Define a Windows Service Monitoring Job

You can define a Windows Service Monitoring (OMS) job to monitor a service on a Windows computer where the agent is running.

Note: To run these jobs, your system requires CA WA Agent for Windows.

To define a Windows Service Monitoring job

1. Insert a job and specify the following attributes in the definition:

   job_type: OMS

   Specifies that the job type is Windows Service Monitoring.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   win_service_name

   Specifies the name of the local Windows service to be monitored.

2. (Optional) Specify optional Windows Service Monitoring attributes:

   - job_class
   - job_terminator
   - monitor_mode
   - win_service_status

3. (Optional) Specify common attributes that apply to all job types.

   The Windows Service Monitoring job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Monitor a Windows Service

This example monitors a Windows service named App Server. The win_service_status attribute is not specified in the job definition, so the job monitors for a RUNNING status by default. The job completes when the service is running.

```
insert_job: oms_job2
job_type: OMS
machine: winagt
win_service_name: App Server
monitor_mode: WAIT
```

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following Windows Service Monitoring job attributes have default values:

monitor_mode

Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

Default: NOW (The job checks for the conditions immediately and completes.)

win_service_status

Specifies the status of the Windows Service to be monitored.

Default: RUNNING (The job checks if the Windows service is running.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor for a PAUSED Status Immediately

The win_service_status attributes in the following job definition overrides the default value.

This example monitors a Windows service named Proc Server. The job checks the status immediately and completes successfully if the service is paused. If the service is not paused, the job fails.

```
insert_job: oms_job1
job_type: OMS
machine: winagt
win_service_name: Proc Server
win_service_status: PAUSED
monitor_mode: NOW
```

Examples: Monitoring Windows Services

The following examples are Windows Services Monitoring Jobs:

Example: Monitor for the Existence of a Windows Service

This example monitors a Windows service named Proc Server. The job completes successfully if the service exists. By default, the job checks for the condition immediately and completes, so the monitor_mode attribute is not required in the job definition.

```
insert_job: oms_job1
job_type: OMS
machine: winagt
win_service_name: Proc Server
win_service_status: EXISTS
```

Example: Specify a Path to a Windows Service Executable

This example monitors a Windows service named Log App. The job waits until the service status is CONTINUE_PENDING before it completes.

```
insert_job: oms_job2
job_type: OMS
machine: winagt
win_service_name: "C:\Program Files\Log App\apptask.exe"
win_service_status: CONTINUE_PENDING
monitor_mode: WAIT
```

Example: Check a Service Status Immediately

This example monitors the schedmanager service for a status of RUNNING. The job checks the status immediately and completes successfully if the service is running. If the service is not running, the job fails.

```
insert_job: oms_job3
job_type: OMS
machine: winagt
win_service_name: schedmanager
win_service_status: RUNNING
monitor_mode: NOW
```

Chapter 14: Oracle E-Business Suite Jobs

This section contains the following topics:

Oracle E-Business Suite Jobs

You can define jobs to run Oracle E-Business Suite workload.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Oracle E-Business Suite.

You can define the following Oracle E-Business Suite jobs:

Copy Single Request Job

Copies an existing single request defined on Oracle Applications and runs it under the agent.

Request Set Job

Runs multiple programs in an Oracle Applications application.

Single Request Job

Runs a single program in an Oracle Applications application.

Define an Oracle E-Business Suite Copy Single Request Job

You can define an Oracle E-Business Suite Copy Single Request (OACOPY) job to copy an existing single request defined on Oracle E-Business Suite and run it under the agent. When the job runs, it can override values in the original definition with values specified on the agent or in the job definition. The OACOPY job is useful when you want to reuse existing job definitions.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Oracle E-Business Suite.

To define an Oracle E-Business Suite Copy Single Request job

1. Insert a job and specify the following attributes in the definition:

   job_type: OACOPY

   Specifies that the job type is Oracle E-Business Suite Copy Single Request.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   request_id

   Specifies the request ID of the Oracle E-Business Suite request you want to copy.

2. Do *one* of the following:

   - Ensure that a default Oracle E-Business Suite user name and responsibility name are defined in the agent's agentparm.txt file using the oa.default.user and oa.default.responsibility parameters, respectively.
   - Add the following attributes to the definition:

     oracle_user

     Specifies the Oracle E-Business Suite user name that the job runs under.

     Note: This attribute overrides the oa.default.user agent parameter.

     oracle_resp

     Specifies an Oracle E-Business Suite responsibility name.

     Note: This attribute overrides the oa.default.responsibility agent parameter.

3. (Optional) Specify optional Oracle E-Business Suite Copy Single Request attributes:

   - job_class
   - job_terminator
   - oracle_mon_children
   - oracle_mon_children_delay

4. (Optional) Specify common attributes that apply to all jobs.

   The Oracle E-Business Suite Copy Single Request job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Copy a Single Request

Suppose that you want to copy an existing single request defined on Oracle E-Business Suite. In this example, the job copies the single request job with request ID 2255470 and overrides the Oracle E-Business Suite user name and responsibility name defined in the agentparm.txt file.

```
insert_job: oacopy_single
job_type: oacopy
machine: oaagent
request_id: 2255470
oracle_user: SYSADMIN
oracle_resp: System Administrator
```

Define an Oracle E-Business Suite Request Set Job

You can define an Oracle E-Business Suite Request Set (OASET) job to run a request set program. You must get the following information from the original Oracle E-Business Suite request set:

- Display name or short name of the Oracle E-Business Suite application the request set belongs to
- Request set name
- User name that the job runs under
- Responsibility name

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Oracle E-Business Suite.

To define an Oracle E-Business Suite Request Set job

1. Insert a job and specify the following attributes in the definition:

   job_type: OASET

   Specifies that the job type is Oracle E-Business Suite Request Set.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   oracle_appl_name

   Specifies the name of the Oracle E-Business Suite application that the request set belongs to.

   oracle_req_set

   Specifies the short name of an Oracle E-Business Suite request set.

2. Do *one* of the following:

   - Ensure that a default Oracle E-Business Suite user name and responsibility name are defined in the agent's agentparm.txt file using the oa.default.user and oa.default.responsibility parameters, respectively.
   - Add the following attributes to the definition:

     oracle_user

     Specifies the Oracle E-Business Suite user name that the job runs under.

     Note: This attribute overrides the oa.default.user agent parameter.

     oracle_resp

     Specifies an Oracle E-Business Suite responsibility name.

     Note: This attribute overrides the oa.default.responsibility agent parameter.

3. (Optional) Specify optional Oracle E-Business Suite Request Set attributes:
   - job_class
   - job_terminator
   - oracle_appl_name_type
   - oracle_mon_children
   - oracle_mon_children_delay
   - oracle_print_copies
   - oracle_print_style
   - oracle_printer
   - oracle_programdata
   - oracle_save_output
   - oracle_use_arg_def
4. (Optional) Specify common attributes that apply to all job types.

   The Oracle E-Business Suite Request Set job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Run a Request Set

This example runs a request set named FNDRSSUB1310. The request set belongs to the application with the short name BIS in Oracle E-Business Suite. The job uses the default Oracle E-Business Suite responsibility name and user name defined on the agent.

```
insert_job: oaset_resp
job_type: oaset
machine: oaagent
oracle_appl_name_type: SHORT
oracle_appl_name: BIS
oracle_req_set: FNDRSSUB1310
```

Specify Data for an Individual Program in a Request Set

When you define an Oracle E-Business Suite Request Set (OASET) job, you can specify data for an individual program in the request set. This data overrides the arguments and print parameters specified for the entire request set. You can specify the following data for a program:

- Program arguments
- Printer
- Print style
- Number of copies to print
- Whether to save the output from the program

To specify data for an individual program in a request set

1. Define an Oracle E-Business Suite Request Set job (see page 351).
2. Add the following attribute to the job definition:

   oracle_programdata

   Specifies data for an individual program in an Oracle E-Business Suite request set.
3. Run the job.

   The data is specified for the program.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Specify Values for Individual Programs in a Request Set

Suppose that you want to run an Oracle E-Business Suite request set named EXTRACTS on the oaagent agent. The request set belongs to the application with the display name Application Object Library in Oracle E-Business Suite. The job definition specifies the following default settings for all programs in the request set:

- The number of copies to be printed is 1.
- The print style is LANDSCAPE.
- The printer is \\printer path\Q8.

The first oracle_programdata attribute overrides the default values for the first program in the request set. The first program uses the arguments T23 and R1 and prints two copies using the \\printer path\Q1 printer in PORTRAIT style.

The second oracle_programdata attribute overrides the default values for the fifth program in the request set. The fifth program uses arguments R and R1 and prints three copies using the \\printer path\Q2 printer in PORTRAIT style.

The other programs in the request set use the default values.

```
insert_job: oaset_prog
job_type: oaset
machine: oaagent
oracle_appl_name_type: DISPLAY
oracle_appl_name: Application Object Library
oracle_req_set: EXTRACTS
oracle_user: SYSADMIN
oracle_resp: System Administrator
oracle_printer: "\\printer path\Q8"
oracle_print_style: LANDSCAPE
oracle_print_copies: 1
oracle_programdata: index=1, args="T23,,R1", printer="\\printer\Q1",
print_style=PORTRAIT, print_copies=2, saveop=Y
oracle_programdata: index=5, args="R,R1,", printer="\\printer\Q2",
print_style=PORTRAIT, print_copies=3, saveop=N
```

Specify Argument Values to Pass to a Program in a Request Set

You can pass argument values to an individual program in an Oracle E-Business Suite Request Set (OASET) job. The job can also use the default values that are defined by the registered Oracle Applications Concurrent Manager program. When an argument value is defined in both the job definition and as a default, the argument value in the job definition overrides the default.

To specify argument values to pass to a program in a request set

1. Define an Oracle E-Business Suite Request Set job (see page 351).
2. Add the following attribute to the job definition:

   oracle_programdata

   Specifies data for an individual program in an Oracle E-Business Suite request set.

3. (Optional) Add the following attribute:

   oracle_use_arg_def

   Specifies whether to use default values for arguments that are not defined using the oracle_programdata attribute.

4. Run the job.

   The argument values are passed to the program in the request set.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Specify Argument Values for a Program in a Request Set

This example runs an Oracle E-Business Suite Request Set job that uses the argument defaults in Oracle E-Business Suite and the argument values defined in the job definition. The second program in the request set has four arguments and you want to pass T23 as the first value and R1 as the fourth value. The argument string specifies placeholders for the second and third arguments, so the job uses the default values for those arguments.

```
insert_job: oaset_prog
job_type: oaset
machine: oaagent
oracle_appl_name_type: DISPLAY
oracle_appl_name: Application Object Library
oracle_req_set: EXTRACTS
oracle_user: SYSADMIN
oracle_resp: System Administrator
oracle_use_arg_def: Y
oracle_programdata: index=2, args="T23,,R1", printer="\\printer\Q1",
print_style=PORTRAIT, print_copies=2, saveop=Y
```

Define an Oracle E-Business Suite Single Request Job

You can define an Oracle E-Business Suite Single Request (OASG) job to run a single request program. You must get the following information from the original Oracle E-Business Suite single request:

- Display name or short name of the Oracle E-Business Suite application the single request belongs to
- Program short name
- User name that the job runs under
- Responsibility name

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Oracle E-Business Suite.

To define an Oracle E-Business Suite Single Request job

1. Insert a job and specify the following attributes in the definition:

   job_type: OASG

   Specifies that the job type is Oracle E-Business Suite Single Request.

   machine

   Specifies the name of an existing machine definition where the agent runs.

oracle_appl_name

Specifies the name of the Oracle E-Business Suite application that the single request belongs to.

oracle_program

Specifies the short name of an Oracle E-Business Suite single request program.

2. Do *one* of the following:
   - Ensure that a default Oracle E-Business Suite user name and responsibility name are defined in the agent's agentparm.txt file using the oa.default.user and oa.default.responsibility parameters, respectively.
   - Add the following attributes to the definition:

     oracle_user

     Specifies the Oracle E-Business Suite user name that the job runs under.

     Note: This attribute overrides the oa.default.user agent parameter.

     oracle_resp

     Specifies an Oracle E-Business Suite responsibility name.

     Note: This attribute overrides the oa.default.responsibility agent parameter.
3. (Optional) Specify optional Oracle E-Business Suite Single Request attributes:
   - job_class
   - job_terminator
   - oracle_appl_name_type
   - oracle_args
   - oracle_desc
   - oracle_mon_children
   - oracle_mon_children_delay
   - oracle_print_copies
   - oracle_print_style
   - oracle_printer
   - oracle_save_output
   - oracle_use_arg_def
4. (Optional) Specify common attributes that apply to all job types.

   The Oracle E-Business Suite Single Request job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Run a Single Request

This example runs a single request program named FNDSCARU. The single request belongs to the application with the short name ACCOUNTS in Oracle E-Business Suite. The job runs under the SYSADMIN user with System Administrator responsibility.

```
insert_job: oasg_short
job_type: oasg
machine: oaagent
oracle_appl_name_type: SHORT
oracle_appl_name: ACCOUNTS
oracle_program: FNDSCARU
oracle_user: SYSADMIN
oracle_resp: System Administrator
```

Specify Argument Values to Pass to a Program in a Single Request

You can pass argument values to a program in an Oracle E-Business Suite Single Request (OASG) job. The job can also use the default values that are defined by the registered Oracle Applications Concurrent Manager program. When an argument value is defined in both the job definition and as a default, the argument value in the job definition overrides the default.

To specify argument values to pass to a program in a single request

1. Define an Oracle E-Business Suite Single Request job (see page 356).
2. Add the following attribute to the job definition:

   oracle_args

   Defines the argument values to pass to an Oracle E-Business Suite single request.

3. (Optional) Add the following attribute:

   oracle_use_arg_def

   Specifies whether to use default values for arguments that are not defined using the oracle_args attribute.

4. Run the job.

   The argument values are passed to the program in the single request.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Specify Argument Values for a Single Request

Suppose that you want to pass the argument values, T,*DefArg2*,X23,,*DefArg5*, to a single request program. The job uses the argument values that are specified in the job definition and the default values defined in Oracle E-Business Suite as follows:

- The first argument, T, and the third argument, X23, are specified in the job definition.
- The second argument, *DefArg2*, and the fifth argument, *DefArg5*, are defined as defaults in Oracle E-Business Suite.
- The fourth argument is not specified in the job definition or defined as a default on Oracle E-Business Suite, so the agent passes an empty string for that argument.

```
insert_job: oasg_args
job_type: oasg
machine: oaagent
oracle_appl_name_type: DISPLAY
oracle_appl_name: Application Object Library
oracle_user: SYSADMIN
oracle_resp: System Administrator
oracle_program: FNDSCARU
oracle_use_arg_def: Y
oracle_args: T,,X23,,
```

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following Oracle E-Business Suite job attributes have default values:

oracle_appl_name_type

Specifies whether the name of the Oracle E-Business Suite application is the short name or the display name.

Default: SHORT

oracle_desc (OASG jobs only)

Defines a description for the Oracle E-Business Suite Single Request job, which is displayed in the Oracle Concurrent Manager.

Default: oa.default.desc agent parameter, if specified

oracle_mon_children

Specifies whether the children of the Oracle E-Business Suite programs are monitored.

Default: N (The job does not monitor children programs.)

oracle_print_style

Specifies an Oracle Applications print style.

Default: oa.default.printStyle agent parameter, if specified

oracle_printer

Specifies the name of a printer to be used by Oracle Applications.

Default: oa.default.printer agent parameter, if specified

oracle_resp

Specifies an Oracle Applications responsibility name. You must also specify an Oracle Applications user name using the oracle_user attribute or by setting the oa.default.user parameter in the agentparm.txt file.

Default: oa.default.responsibility agent parameter, if specified

Note: All Oracle Applications jobs require a responsibility name. If you do not specify this attribute in the job definition, a default responsibility name must be defined in the agent's agentparm.txt file using the oa.default.responsibility parameter. Otherwise, the job fails.

oracle_save_output (OASG and OASET jobs only)

Specifies whether to save the output from an Oracle E-Business Suite Single Request or Request Set job.

Default: N (The job does not save the output.)

oracle_use_arg_def (OASG and OASET jobs only)

Specifies whether to use default values for arguments that not defined using the oracle_args attribute or the oracle_programdata attribute. The default arguments are defined in Oracle E-Business Suite.

Default: N (The job does not use the default values for the arguments.)

oracle_user

Specifies an Oracle Applications user name that the job runs under.

Default: oa.default.user agent parameter, if specified

Note: All Oracle Applications jobs require a user name. If you do not specify this attribute in the job definition, a default user must be defined in the agent's agentparm.txt file using the oa.default.user parameter. Otherwise, the job fails.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Override a Default Value in an OASET Job

This example overrides the default responsibility name using the oracle_resp attribute. The job also overrides the default user that the job runs under using the oracle_user attribute.

```
insert_job: oaset_resp
job_type: oaset
machine: oaagent
oracle_appl_name_type: SHORT
oracle_appl_name: BIS
oracle_req_set: FNDRSSUB1310
oracle_user: SYSADMIN
oracle_resp: System Administrator
```

Chapter 15: PeopleSoft Jobs

This section contains the following topics:

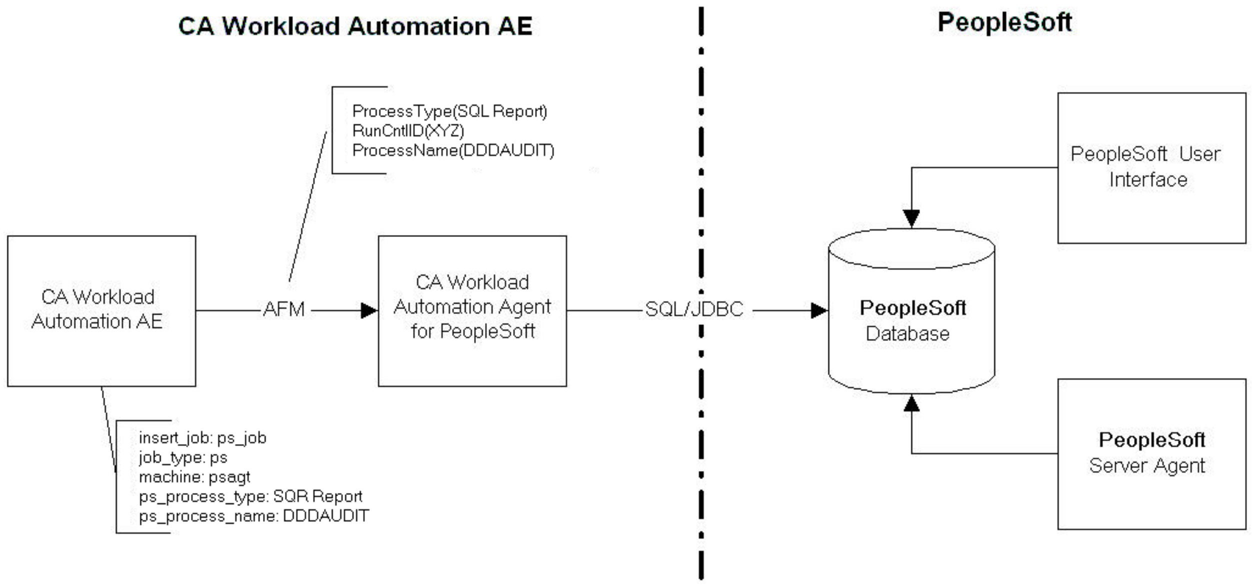
PeopleSoft Jobs

PeopleSoft jobs let you run different types of PeopleSoft processes defined in your PeopleSoft system. For example, you can define PeopleSoft jobs to execute PeopleSoft programs and report the program status.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for PeopleSoft.

When you define a PeopleSoft job, you can set the output type and format of a report. For email and web output types, you can set various distribution properties such as the recipients and message text. You can also pass run control parameter values that will be stored in the corresponding run control table.

When a PeopleSoft program runs, it modifies its run status (RUNSTATUS) in the PSPRCSRQST table in the PS database. The following diagram shows the functional relationship between the scheduling manager, the agent, and the PeopleSoft system:


CA Workload Automation AE
PeopleSoft
ProcessType(SQL Report)
RunCntlID(XYZ)
ProcessName(DDDAUDIT)
CA Workload Automation AE
AFM
CA Workload Automation Agent for PeopleSoft
SQL/JDBC
PeopleSoft Database
PeopleSoft User Interface
PeopleSoft Server Agent
insert_job: ps_job
job_type: ps
machine: psagt
ps_process_type: SQR Report
ps_process_name: DDDAUDIT

PeopleSoft Exit Codes

A PeopleSoft (PS) job can return exit codes 9 and 17, which indicate success. When a PS job terminates with either of these exit codes, CA Workload Automation AE changes the status of the job to SUCCESS and logs the non-zero exit code.

PeopleSoft User IDs and Passwords

The operator ID sets the authority for running PeopleSoft reports. Your agent administrator can set defaults for an operator ID and corresponding password using the ps.default.oprId and ps.default.oprPassword parameters in the agentparm.txt. You can override the default operator ID by specifying the ps_operator_id attribute in a PeopleSoft job definition. The ps_operator_id must be defined on CA Workload Automation AE using the autosys_secure command unless the ps.skipOprPswdValidation=true parameter is configured in the agentparm.txt file.

Note: For more information about the autosys_secure command, see the *Reference Guide*.

Define a PeopleSoft Job

You can define a PeopleSoft (PS) job to schedule workload to run in PeopleSoft. The job runs a PeopleSoft process request or a collection of process requests.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for PeopleSoft.

To define a PeopleSoft job

1. Insert a job and specify the following attributes in the definition:

   job_type: PS

   Specifies that the job type is PeopleSoft.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   ps_process_name

   Specifies the name of the PeopleSoft report to run. This value corresponds to the Process Name field in PeopleSoft.

   ps_process_type

   Specifies the type of PeopleSoft report that you want the job to run.

2. Do *one* of the following:

   - Ensure that a default run control ID is defined in the agent's agentparm.txt file using the ps.default.runCntlId parameter.
   - Add the following attribute to the definition:

     ps_run_cntrl_id

     Specifies the value assigned to the run control identifier. This value corresponds to the Run Control ID field in PeopleSoft.

     Note: This attribute overrides the ps.default.runCntlId agent parameter.

3. (Optional) Specify optional PeopleSoft attributes:

   - envvars
   - job_class
   - job_terminator
   - ps_args
   - ps_dest_format
   - ps_dest_type
   - ps_detail_folder

- ps_dlist_roles
- ps_dlist_users
- ps_email_address
- ps_email_address_expanded
- ps_email_log
- ps_email_subject
- ps_email_text
- ps_email_web_report
- ps_operator_id
- ps_output_dest
- ps_restarts
- ps_run_cntrl_args
- ps_run_control_table
- ps_server_name
- ps_skip_parm_updates
- ps_time_zone

Note: A PeopleSoft job must have a valid operator ID to run successfully. Check with your agent administrator to determine whether a default operator ID is set on the agent. If a default is not set, then you must specify the ps_operator_id attribute in a job definition.

4. (Optional) Specify common attributes that apply to all jobs.

   The PeopleSoft job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Run a PeopleSoft Process

This example runs an Application Engine process named DDDAUDIT. The job runs on the agent named psagt.

```
insert_job: ps_txtfile
job_type: ps
machine: psagt
ps_process_name: DDDAUDIT
ps_process_type: Application Engine
```

Note: The job uses the default operator ID, output destination format, output destination type, and run control ID defined on the agent. The PeopleSoft Server that runs the job is not defined in the job definition or as a default on the agent, so the PeopleSoft Process Scheduler determines the PeopleSoft Server that will run the job.

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following PeopleSoft job attributes have default values:

ps_dest_format

Specifies the type of format for the report output.

Default: ps.default.outDestFormat agent parameter, if specified

ps_dest_type

Specifies the output destination type for the PeopleSoft report.

Default: ps.default.outDestType agent parameter, if specified

ps_email_log

Specifies whether to email job logs with the PeopleSoft report to recipients on a distribution list.

Default: No (Job logs are not emailed to recipients.)

ps_email_web_report

Specifies whether to email a web report to recipients on a distribution list.

Default: No (A web report is not emailed to recipients.)

ps_operator_id

Specifies the operator ID under whose authority the PeopleSoft reports run.

Default: ps.default.oprId agent parameter, if specified

ps_output_dest

Specifies the output destination for the PeopleSoft request. The destination can be a file directory or a printer.

Default: If ps_dest_type is PRINTER, the default is one of the following, in the following order:

- ps.default.printer parameter in the agent's agentparm.txt file, if specified
- The default PeopleSoft printer, lpt1

ps_restarts

Specifies whether to disable a restart feature for previously failed jobs from the point where the job failed.

Default: No (The restart feature is not disabled.)

ps_run_cntrl_id

Specifies a set of PeopleSoft run parameters for a given PeopleSoft process.

Default: ps.default.runCntlId agent parameter, if specified

Note: All PeopleSoft jobs require a run control ID. If you do not specify this attribute in the job definition, a default run control ID must be defined in the ps.default.runCntlId parameter in the agent's agentparm.txt file. Otherwise, the job fails.

ps_server_name

Specifies the target server that runs the PeopleSoft job.

Default: ps.default.serverName

Note: If the PeopleSoft Server is not specified as a default on the agent or in the job definition, the PeopleSoft Process Scheduler determines the PeopleSoft Server that will run the job.

ps_skip_parm_updates

Specifies whether you want the agent to update job parameters with data in the PS_PRCSDEFN table.

Default: NO (The agent updates job parameters with data in the PS_PRCSDEFN table.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Specify a Run Control ID and a Server Name

This example overrides the default run control ID using the ps_run_cntrl_id attribute and server name using the ps_server_name attribute. The job uses the Application Engine process type. The job uses the default output destination format and type defined on the agent. The job overrides the default operator ID using the ps_operator_id attribute.

```
insert_job: ps_cst
job_type: ps
machine: psagt
ps_process_name: PAYROLL
ps_server_name: PSPR
ps_process_type: Application Engine
ps_run_cntrl_id: PS_ALL
ps_operator_id: vp1@ps1
```

Mapping of PeopleSoft Fields to Job Attributes

When you define a PeopleSoft job, you specify JIL attributes that map to your PeopleSoft process request. The following table maps the PeopleSoft fields to the attributes:

PeopleSoft Field Name	Attribute Name
Format	ps_dest_format
Type	ps_dest_type
Folder Name	ps_detail_folder
ID Type (Role selected) Distribution ID	ps_dlist_roles
ID Type (User selected) Distribution ID	ps_dlist_users
Email Address List	ps_email_address
Email With Log	ps_email_log
Email Subject	ps_email_subject
Message Text	ps_email_text
Email Web Report	ps_email_web_report
Output Destination	ps_output_dest
Process Name	ps_process_name

PeopleSoft Field Name	Attribute Name
Process Type	ps_process_type
Run Control Arguments	ps_run_cntrl_args
Run Control ID	ps_run_cntrl_id
Run Control Table	ps_run_control_table
Server Name	ps_server_name
Time Zone	ps_time_zone

Distribute a PeopleSoft Report

If you specify EMAIL as the output destination type (ps_dest_type), you can distribute a PeopleSoft report electronically to operators, groups of people, or individuals.

To distribute a PeopleSoft report

1. Define a PeopleSoft job (see page 365).
2. Add the following attributes to the job definition:

   ps_dest_type: EMAIL

   Sends the output of the PeopleSoft report as an email message. This attribute corresponds to the Type field in PeopleSoft.

   ps_dest_format

   Specifies the field name of the output destination format. PeopleSoft stores the list of output destination formats in the PSXLATITEM table. This value corresponds to the Format field in PeopleSoft.

3. Add *one or both* of the following attributes:

   ps_dlist_roles

   Specifies a distribution list of the roles that represent the individuals who are receiving the PeopleSoft report. This value corresponds to the ID Type field (with Role selected) and the Distribution ID field in PeopleSoft.

   ps_dlist_users

   Specifies a distribution list of operator IDs to send a PeopleSoft report to. This value corresponds to the ID Type field (with User selected) and the Distribution ID field in PeopleSoft.

4. (Optional) Add the following attributes:

   ps_email_address

   Specifies the email addresses of the recipients on a distribution list. This value corresponds to the Email Address List field in PeopleSoft.

   ps_email_subject

   Defines an email subject to include in the email. This value corresponds to the Email Subject field in PeopleSoft.

   ps_email_text

   Defines the body text of the email. This value corresponds to the Message Text field in PeopleSoft.

5. Run the job.

   The PeopleSoft report is sent to the specified distribution lists and email addresses.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Distribute a Report to Users

This example runs a Crystal report under the VP3 operator ID. The report is formatted as PDF and distributed in an email to the VP1, VP2, and VP3 operator IDs.

```
insert_job: ps_users
job_type: PS
machine: psagt
ps_process_name: XRFWIN
ps_process_type: Crystal
ps_dest_type: EMAIL
ps_dest_format: PDF
ps_dlist_users: VP1,VP2,VP3
ps_operator_id: VP3@ps1
ps_run_cntrl_id: test
```

Example: Email a PeopleSoft Report

This example runs a Crystal report and emails the output to recipients. The Crystal report runs under the VP2 operator ID. The output is sent to the email addresses specified in the ps_email_address attribute. The email includes a subject title.

```
insert_job: ps_email
job_type: PS
machine: psagt
ps_process_name: XRFWIN
ps_process_type: Crystal
ps_dest_type: EMAIL
ps_dest_format: PDF
ps_operator_id: VP2@ps1
ps_email_address: user1@example.com;user2@example.com
ps_email_subject: PeopleSoft Report Status
ps_email_text: This report is available for distribution.
```

Store the Output of a PeopleSoft Job as a Web Report

You can define a PeopleSoft job to run a process and store the output as a web report to view later. You can also define the job to email the web report to one or more recipients.

To store the output of a PeopleSoft job as a web report

1. Define a PeopleSoft job (see page 365).
2. Add the following attributes to the job definition:

   ps_dest_type: WEB

   Posts the output of the PeopleSoft report on a website. This attribute corresponds to the Type field in PeopleSoft.

   ps_dest_format

   Specifies the field name of the output destination format. PeopleSoft stores the list of output destination formats in the PSXLATITEM table. This value corresponds to the Format field in PeopleSoft.

3. (Optional) Add the following attributes:

   ps_email_web_report: YES

   Specifies that the job emails a web report to the recipients on the distribution list. This attribute corresponds to the Email Web Report field in PeopleSoft.

   ps_email_address

   Specifies the email addresses of the recipients on a distribution list. This value corresponds to the Email Address List field in PeopleSoft.

ps_email_subject

Defines an email subject to include in the email. This value corresponds to the Email Subject field in PeopleSoft.

ps_email_text

Defines the body text of the email. This value corresponds to the Message Text field in PeopleSoft.

4. Run the job.

   The output of the PeopleSoft job is stored as a web report. The job emails the web report to recipients if the email attributes are specified.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Format a PeopleSoft Job Output as an HTML Web Report

This example runs the XRFWIN process. The process type is SQR Report and the run control ID is PS_ALL. The server named PSPR runs the job, and the output is stored as an HTML web report.

```
insert_job: ps_htmfile
job_type: ps
machine: psagt
ps_process_name: XRFWIN
ps_process_type: SQR Report
ps_server_name: PSPR
ps_dest_type: WEB
ps_dest_format: HTM
ps_run_cntrl_id: PS_ALL
```

Example: Distribute a PeopleSoft Web Report in an Email

This example stores the output as a PDF web report. The web report is sent to the email addresses specified in the ps_email_address attribute. The email includes a subject title and body text.

```
insert_job: ps_web
job_type: PS
machine: psagt
ps_process_name: XRFWIN
ps_process_type: Crystal
ps_dest_type: WEB
ps_dest_format: PDF
ps_operator_id: VP2@ps1
ps_email_web_report: YES
ps_email_address: user1@example.com;user2@example.com
ps_email_subject: PeopleSoft Report Status
ps_email_text: This report is available for distribution.
```

Send the Output of a PeopleSoft Job to a Printer

You can define a PeopleSoft job to run a process and send the output to a printer.

To send the output of a PeopleSoft job to a printer

1. Define a PeopleSoft job (see page 365).
2. Add the following attributes to the job definition:

   ps_dest_type: PRINTER

   Sends the output of the PeopleSoft report to a printer. This attribute corresponds to the Type field in PeopleSoft.

   ps_output_dest

   Specifies the network location of the printer including the printer server and shared printer name. This value corresponds to the Output Destination field in PeopleSoft.
3. Run the job.

   The output of the PeopleSoft job is sent to a printer.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Send a Job's Output to a Specified Printer

This example runs an SQR Report. The report is formatted as PS and outputted to a printer.

```
insert_job: ps_printer
job_type: ps
machine: psagent
ps_process_name: XRFWIN
ps_process_type: SQR Report
ps_dest_type: PRINTER
ps_dest_format: PS
ps_output_dest: \\printers\PRINTER1
ps_run_cntrl_id: test
ps_operator_id: VP1@ps1
```

Chapter 16: SAP Jobs

This section contains the following topics:

SAP Jobs

SAP jobs let you run SAP workload.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for SAP.

You can define the following SAP jobs:

SAP Batch Input Session

Imports data from external systems to the SAP system.

SAP Business Warehouse (BW) InfoPackage

Transfers data from a data source to an SAP Business Warehouse system.

SAP Business Warehouse (BW) Process Chain

Creates Process Chains on the SAP system.

SAP Data Archiving

Stores information in an SAP Archiving Object.

SAP Event Monitor

Monitors and triggers SAP events.

SAP Job Copy

Copies an existing SAP R/3 job.

SAP Process Monitor

Monitors for a specific SAP process status.

SAP R/3

Schedules an SAP R/3 job on your SAP system.

SAP Connection Attributes

The following connection attributes are common to SAP job types and may be required in your job definitions:

sap_rfc_dest

Specifies the SAP system to connect to. This value corresponds to the destination@properties agent configuration file name containing the SAP connection information. The sap_rfc_dest value overrides the sap.default.destination configuration value in the agentparm.txt file.

sap_client

Specifies the client number associated with an SAP instance. An SAP instance can have multiple clients defined for it. Each client has its own data. Your agent administrator can define a default SAP client number using the jco.client.client property in the connection properties file.

sap_lang

Specifies the language required to run a job. Your agent administrator can define a default language using the jco.client.lang property in the connection properties file.

sap_target_sys

Specifies the host name of an SAP application server where the job is to run.

Note: If the sap_target_sys attribute is not defined, the SAP system will select an application server based on available resources.

Notes:

- If a default value is not defined on the agent, you must specify the attribute in the job definition.
- For more information about these attributes, see the *Reference Guide*.

SAP User IDs and Passwords

All SAP user IDs and passwords must be defined on CA Workload Automation AE by using the autosys_secure command. When you define an SAP job, specify an SAP user ID using the owner attribute, or use the default owner value. The job runs under this user ID.

Note: For more information about the autosys_secure command, see the *Reference Guide*.

Define an SAP Batch Input Session Job

You can define an SAP Batch Input Session (SAPBDC) job to import large amounts of data from external systems to the SAP system.

To schedule a BDC job, you first define an ABAP that creates a Batch Input Session (BDC ABAP) on the SAP system. Next, you schedule an SAP Batch Input Session job in CA Workload Automation AE to run the BDC ABAP on the SAP system. After the job runs, the BDC job starts the data transfer.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for SAP.

To define an SAP Batch Input Session job

1. Insert a job and specify the following attributes in the definition:

   job_type: SAPBDC

   Specifies that the job type is SAP Batch Input Session.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   sap_job_name

   Specifies the name of the ABAP job that creates the BDC session on the SAP system.

   sap_step_parms

   Defines the SAP R/3 step specifications.

   Note: We recommend that you limit the number of steps (ABAPs) to one per job. If you run a job and one of the ABAPs fails, the job is marked as failed. If the ABAP fails, you cannot re-run the ABAP without re-running the entire job.

2. (Optional) Specify optional SAP Batch Input Session attributes:
   - bdc_err_rate
   - bdc_ext_log
   - bdc_proc_rate
   - bdc_system
   - job_class
   - job_terminator
   - sap_client
   - sap_job_class
   - sap_lang
   - sap_office
   - sap_recipients
   - sap_release_option
   - sap_rfc_dest
   - sap_step_parms
   - sap_target_sys
3. (Optional) Specify common attributes that apply to all job types.

   The SAP Batch Input Session job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Define an SAP Batch Input Session Job

This example runs the ZBDCTEST ABAP that creates a Batch Input Session (BDC ABAP) on the default SAP system defined to the agent. The job runs as soon as possible after the job is defined. After this job runs, the BDC job starts the data transfer.

```
insert_job: bdcjob
job_type: SAPBDC
machine: sapagent
owner: WAAESAP@sapagent
sap_job_name: ZBDCTEST
sap_release_option: A
sap_step_parms: abap_name="ZBDCTEST"
```

Example: Define an SAP Batch Input Session Job with Step Parameters

This example runs the SAP Batch Input Session job named ZBDCTEST. When SAPBDC_job runs, it releases ZBDCTEST immediately on the SAP system. If no free background processing is available, the ZBDCTEST is not released and stays in the Scheduled SAP job state.

```
insert_job: SAPBDC_job
job_type: SAPBDC
machine: localhost
owner: WAAESAP@sapserver
sap_job_name: ZBDCTEST
sap_release_option: I
sap_step_parms: abap_name="ZBDCTEST",banner_page=no,release=no,print_imm=no,
new_spool=no,footer=no
```

Define an SAP BW InfoPackage Job

You can define an SAP BW InfoPackage (SAPBWIP) job to transfer data from any data source into an SAP Business Warehouse system. When the job runs, the data is transferred.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for SAP.

To define an SAP BW InfoPackage job

1. Insert a job and specify the following attributes in the definition:

   job_type: SAPBWIP

   Specifies that the job type is SAP BW InfoPackage.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   sap_info_pack

   Specifies the name of the Business Warehouse InfoPackage.

2. (Optional) Specify optional SAP BW InfoPackage attributes:

   - job_class
   - job_terminator
   - sap_client
   - sap_ext_table
   - sap_job_name
   - sap_lang
   - sap_rfc_dest

3. (Optional) Specify common attributes that apply to all job types.

   The SAP BW InfoPackage job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Define an SAP BW InfoPackage Job

This example runs the Business Warehouse InfoPackage 0PAK_D2XZMZ1HD5WFVFL3EN1NVIT4V at the SAP destination, SM1.

```
insert_job: InfoBW2
job_type: SAPBWIP
machine: sapagent
owner: WAAESAP@sapagent
sap_job_name: InfoBW2
sap_rfc_dest: SM1
sap_info_pack: 0PAK_D2XZMZ1HD5WFVFL3EN1NVIT4V
```

Example: Define an SAP BW InfoPackage Job

This example runs the Business Warehouse InfoPackage ZPAK_DIBX41NKYK0S7FB1FOV0FQ7BV on the SAP system.

```
insert_job: bwip_job
job_type: SAPBWIP
machine: localhost
owner: user@sapserver
sap_job_name: BI_BTCHSAP_TEST
sap_rfc_dest: mts
sap_info_pack: ZPAK_DIBX41NKYK0S7FB1FOV0FQ7BV
sap_lang: EN
sap_client: 001
job_class: a
```

Define an SAP BW Process Chain Job

You can define an SAP BW Process Chain (SAPBWPC) job to run a sequence of background processes on the SAP system. Some SAP processes trigger events that can start other processes. An SAPBWPC job runs the individual processes in the chain as job steps.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for SAP.

To define an SAP BW Process Chain job

1. Insert a job and specify the following attributes in the definition:

   job_type: SAPBWPC

   Specifies that the job type is SAP BW Process Chain.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   sap_chain_id

   Specifies the name of the Business Warehouse Process Chain.

2. (Optional) Specify optional SAP BW Process Chain attributes:
   - job_class
   - job_terminator
   - sap_client
   - sap_lang
   - sap_rfc_dest
3. (Optional) Specify common attributes that apply to all job types.

   The SAP BW Process Chain job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Define an SAP BW Process Chain Job

This example defines the SAPBWPC3 job that runs the DEMO_CHAIN1 Process Chain on the SAP system. The language used to log on to the SAP system is English.

```
insert_job: SAPBWPC3
job_type: SAPBWPC
machine: sapagent
owner: WAAESAP@sapagent
sap_chain_id: DEMO_CHAIN1
sap_lang: EN
sap_rfc_dest: SM1
```

Define an SAP Data Archiving Job

You can define an SAP Data Archiving (SAPDA) job to store information described in an SAP Archiving Object into an SAP data archive.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for SAP.

To define an SAP Data Archiving job

1. Insert a job and specify the following attributes in the definition:

   job_type: SAPDA

   Specifies that the job type is SAP Data Archiving.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   arc_obj_name

   Specifies the name of the archiving object.

   arc_obj_variant

   Specifies the name of the archiving object variant.

2. (Optional) Specify optional SAP Data Archiving attributes:
   - arc_parms
   - job_class
   - job_terminator
   - sap_client
   - sap_lang
   - sap_print_parms
   - sap_rfc_dest
   - sap_target_sys
3. (Optional) Specify common attributes that apply to all job types.

   The SAP Data Archiving job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Define an SAP Data Archiving Job

This example defines a job that stores information described in the BC_ARCHIVE Archiving Object into an SAP data archive. The archiving object variant is BC_ARCVARIANT.

```
insert_job: SAPDA_job
job_type: SAPDA
machine: sapagent
owner: WAAESAP@sapagent
arc_obj_name: BC_ARCHIVE
arc_obj_variant: BC_ARCVARIANT
sap_print_parms: dest=LP01,prt_arc_mode=PRINT
```

Define an SAP Event Monitor Job

You can define an SAP Event Monitor (SAPEVT) job to schedule workload based on the activity of an SAP event or trigger an SAP event at the appropriate time in your schedule.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for SAP.

To define an SAP Event Monitor job

1. Insert a job and specify the following attributes in the definition:

   job_type: SAPEVT

   Specifies that the job type is SAP Event Monitor.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   sap_event_id

   Specifies the name of the SAP event to monitor or trigger.

2. (Optional) Specify optional SAP Event Monitor attributes:
   - continuous
   - job_class
   - job_terminator
   - sap_client
   - sap_event_parm
   - sap_is_trigger
   - sap_lang
   - sap_rfc_dest
3. (Optional) Specify common attributes that apply to all job types.

   The SAP Event Monitor job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Trigger an SAP Event

This example defines an SAPEVT job. The job triggers the SAP_TEST event and the SAP job dependencies waiting on the SAP_TEST event are satisfied.

```
insert_job: SAPEVT_job
job_type: SAPEVT
machine: sapagent
owner: WAAESAP@sapagent
sap_event_id: SAP_TEST
sap_rfc_dest: BI1
sap_is_trigger: Y
```

Example: Monitor an SAP Event

This example defines an SAPEVT job. The sap_is_trigger attribute is set to N, so the job monitors the SAP_TEST event and remains in RUNNING status.

```
insert_job: trigger_evt
job_type: SAPEVT
machine: localhost
owner: WAAESAP@sapserver
sap_rfc_dest: BI1
sap_event_id: SAP_TEST
sap_is_trigger: N
sap_event_parm: L L
sap_client: 001
sap_lang: EN
continuous: y
```

Define an SAP Process Monitor Job

You can define an SAP Process Monitor (SAPPM) job to monitor for a specific SAP process status and end after detecting a process. You can also use SAP Process Monitor jobs to set up predecessor or dependent job relationships with other jobs or SAP processes.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for SAP.

To define an SAP Process Monitor job

1. Insert a job and specify the following attributes in the definition:

   job_type: SAPPM

   Specifies that the job type is SAP Process Monitor.

machine

Specifies the name of an existing machine definition where the agent runs.

sap_process_status

Specifies the SAP process status to monitor (RUNNING, STOPPED, or WAITING).

2. (Optional) Specify the following optional attributes if sap_process_status is set to RUNNING or STOPPED:
   - sap_abap_name
   - sap_proc_user
   - sap_process_client
3. (Optional) Specify optional SAP Process Monitor attributes:
   - continuous
   - job_class
   - job_terminator
   - sap_client
   - sap_lang
   - sap_proc_type
   - sap_rfc_dest
   - sap_target_sys
4. (Optional) Specify common attributes that apply to all job types.

   The SAP Process Monitor job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Define an SAP Process Monitor Job

This example defines an SAP Process Monitor job that monitors for the ABAP program ZMYABAP to change to the RUNNING status.

```
insert_job: test_SAPPM
job_type: SAPPM
machine: sapagent
sap_abap_name: ZMYABAP
sap_process_status: RUNNING
sap_rfc_dest: BI1
owner: WAAESAP@sapagent
```

Define an SAP Job Copy Job

You can define an SAP Job Copy (SAPJC) job to copy an existing SAP R/3 job.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for SAP.

To define an SAP Job Copy job

1. Insert a job and specify the following attributes in the definition:

   job_type: SAPJC

   Specifies that the job type is SAP Job Copy.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   sap_job_count

   Specifies the ID of the job to be copied.

   sap_job_name

   Specifies the name of the SAP R/3 job to be copied.

2. (Optional) Specify optional SAP Job Copy attributes:
   - job_class
   - job_terminator
   - sap_client
   - sap_fail_msg
   - sap_lang
   - sap_mon_child
   - sap_release_option
   - sap_rfc_dest
   - sap_step_num
   - sap_success_msg
   - sap_target_jobname
   - sap_target_sys
3. (Optional) Specify common attributes that apply to all job types.

   The SAP Job Copy job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Define an SAP Job Copy Job

This example defines an SAP Job Copy job that copies the AM job with job count 11331500 and runs the new copy.

```
insert_job: SAPJC_job
job_type: SAPJC
owner: WAAESAP@sapagent
machine: sapagent
sap_job_name: AM
sap_job_count: 11331500
```

Define an SAP R/3 Job

You can define an SAP R/3 job (SAP) to schedule an SAP R/3 job on your SAP system.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for SAP.

To define an SAP R/3 job

1. Insert a job and specify the following attributes in the definition:

   job_type: SAP

   Specifies that the job type is SAP R/3.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   sap_job_name

   Specifies the SAP job name that identifies the workload in the SAP system.

   sap_step_parms

   Defines the SAP R/3 step specifications.

   Note: We recommend that you limit the number of steps (ABAPs) to one per job. If you run a job and one of the ABAPs fails, the job is marked as failed. If the ABAP fails, you cannot re-run the ABAP without re-running the entire job.

2. (Optional) Specify optional SAP R/3 attributes:

   - job_class
   - job_terminator
   - sap_client
   - sap_fail_msg
   - sap_job_class
   - sap_lang
   - sap_mon_child
   - sap_office
   - sap_recipients
   - sap_release_option
   - sap_rfc_dest
   - sap_success_msg
   - sap_target_sys

3. (Optional) Specify common attributes that apply to all job types.

   The SAP R/3 job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Define an SAP R/3 Job With One Step

This example runs the SAP R/3 job named BI_WRITE_PROT_TO_APPLLOG. The job runs the SAP program (ABAP) named RSBATCH_WRITE_PROT_TO_APPLLOG. The owner is defined in CA Workload Automation AE using the autosys_secure command.

```
insert_job: test_SAP
job_type: SAP
machine: sapagent
sap_job_name: BI_WRITE_PROT_TO_APPLLOG
sap_rfc_dest: BI1
sap_step_parms: abap_lang=EN,abap_name=RSBATCH_WRITE_PROT_TO_APPLLOG
owner: WAAESAP@sapagent
```

Example: Define an SAP R/3 Job With Multiple Steps

This example runs an SAP R/3 job named SAP 3 STEPS_@#$. The job runs three steps. Each step runs the same ABAP but with different parameters.

```
insert_job: TEST_1
job_type: SAP
machine: localhost
job_class: c
sap_job_name: "SAP 3 STEPS_@#$"
sap_release_option: I
sap_step_parms:abap_name=BTCTEST,variant=test,arc_printer=LP01,copies=2,prt_arc_m
ode=BOTH,arc_obj_type=ARCHIVE,arc_doc_type=ARCHIVE,arc_info=inf,num_lines=65,num_
columns=80
sap_step_parms:abap_name=BTCTEST,variant=test2,arc_printer=LP01,copies=3,prt_arc_
mode=ARCHIVE,print_imm=N,release=N,sap_banner=Y,banner_page=Y,recipient_name=cybe
r,num_lines=65,num_columns=80,arc_obj_type=ARCHIVE,arc_doc_type=ARCHIVE,arc_info=
inf
sap_step_parms:abap_name=BTCTEST,variant=test,arc_printer=LP01,copies=1,prt_arc_m
ode=ARCHIVE,sap_banner=Y,banner_page=Y,recipient_name=cyber,num_lines=65,num_colu
mns=80,authorization=string2,arc_obj_type=ARCHIVE,arc_doc_type=ARCHIVE,arc_info=i
nf
```

Attributes with Default Values

Some attributes have default values that automatically apply to all job definitions. Your agent administrator can also define default values in the agent's agentparm.txt file or the SAP agent connection properties file. If a default value exists, you do not have to specify the corresponding attribute in the job definition. However, you can specify the attribute to override the default.

The following SAP job attributes have default values:

bdc_ext_log (SAPBDC jobs only)

Specifies whether to generate advanced logging of the Batch Input Session (BDC) running on the SAP system.

Default: N (The job does not generate advanced logging.)

sap_client

Specifies the SAP client within the SAP system.

Default: jco.client.client connection properties parameter, if specified

sap_is_trigger (SAPEVT jobs only)

Specifies whether to trigger or monitor an SAP event.

Default: N (The job monitors an SAP event.)

sap_lang

Specifies a character code representing a valid language for SAP.

Default:

- EN (The default language is English.)
- jco.client.lang connection properties parameter, if specified. This parameter overrides the default type (EN).

sap_mon_child (SAPJC and SAP jobs only)

Specifies whether to monitor children jobs.

Default: N (The job does not monitor children jobs.)

sap_office

Specifies whether to save outgoing documents to the SAPoffice outbox of the SAP user associated with the job.

Default: N (The job does not save outgoing documents.)

sap_release_option (SAPBDC, SAPJC, and SAP jobs only)

Specifies the action to take with a job after it is defined.

Default: A (The job releases the job as soon as possible.)

sap_rfc_dest

Specifies the destination value for the Remote Function Call (RFC) connection and gateway information.

Default: sap.default.destination agent parameter, if specified

sap_target_jobname (SAPJC jobs only)

Specifies the name of the target job to copy.

Default: The name of the source job

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor an SAP Event and Override the Default Client ID

The sap_client attribute in the following job definition overrides the default client specified in the agent connection properties file.

This example defines an SAPEVT job. The sap_is_trigger attribute is set to N, so the job monitors the SAP_TEST event and remains in RUNNING status.

```
insert_job: trigger_evt
job_type: SAPEVT
machine: localhost
owner: WAAESAP@sapserver
sap_rfc_dest: BI1
sap_event_id: SAP_TEST
sap_is_trigger: N
sap_event_parm: L L
sap_client: 001
sap_lang: EN
continuous: y
```

Email an SAP Job's Spool File

You can email the SAP spool file for all steps in an SAP R/3 (SAP) job to recipients. A copy of the spool file is emailed on job completion or failure. The recipient can be an email address, an SAPoffice distribution list, or an SAP user.

To email an SAP job's spool file

1. Define an SAP R/3 job (see page 390).
2. Add the following attribute to the job definition:

   sap_step_parms: abap_name=*abap*, sap_maillist=*value*

   Specifies one or more recipients on a distribution list and the ABAP name.

3. (Optional) Add one or both of the following attributes:

   sap_fail_msg

   Specifies a string that indicates the job failed. If the string is found in the job's spool file, the job is considered failed even if the job succeeds on the SAP system.

   sap_success_msg

   Specifies a string that indicates the job completed successfully. If the string is found in the job's spool file, the job is considered successfully completed even if the job fails on the SAP system.

4. (Optional) Add the following attribute:

   sap_office

   Specifies whether to save outgoing documents to the SAPoffice outbox of the SAP user associated with the job.

5. Run the job.

   The job's spool file is emailed to the recipients.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Email an SAP Job's Spool File

This example emails the SAP spool file for an SAP job to user1@example.com.

```
insert_job: test_SAP_1
job_type: SAP
machine: sapagent
owner: WAAESAP@sapserver
sap_job_name: BI_WRITE_PROT_TO_APPLLOG
sap_rfc_dest: BI1
sap_step_parms:abap_name=RSBATCH_WRITE_PROT_TO_APPLLOG,sap_maillist="user1@exampl
e.com",variant=TEST,banner_page=no,copies=1,release=no,recipient_name=cybermation
,prt_arc_mode=PRINT,print_imm=no,num_lines=65,num_columns=80,new_spool=no,footer=
no
```

Email the Spool File of a Single Step in an SAP Job

You can email the SAP spool file for a single step in an SAP R/3 (SAP) or SAP Batch Input Session (SAPBDC) job to recipients. A copy of the spool file is emailed on step completion or failure. The recipient can be an email address, an SAPoffice distribution list, or an SAP user.

To email the spool file of a single step in an SAP job

1. Define an SAP R/3 job (see page 390) or SAP Batch Input Session job (see page 377).
2. Add the following attribute to the job definition:

```
sap_step_parms: abap_name=abap,sap_maillist=address
                [,sap_fail_msg=message]
                [,sap_success_msg=message]
```

   abap_name=*abap*

   Specifies the valid SAP system ABAP name. This keyword corresponds to the SAPGUI ABAP Program Name field on the Create Step dialog.

sap_maillist=*address*

Specifies one or more recipients to send the spool list results to.

sap_fail_msg=*message*

(Optional) Specifies a string that indicates the failure of the step. If the string matches the SAP ABAP output for the step, the step is considered failed even if the step succeeds on the SAP system.

Note: This keyword does not apply to SAPBDC jobs.

sap_success_msg=*message*

(Optional) Specifies a string that indicates the success of the step. If the string matches the SAP ABAP output for the step, the step is considered successfully completed even if the step fails on the SAP system.

Note: This keyword does not apply to SAPBDC jobs.

3. (Optional) Add optional parameters to the sap_step_parms attribute.
4. (Optional) Add the following attribute:

   sap_office

   Specifies whether to save outgoing documents to the SAPoffice outbox of the SAP user associated with the job.
5. Run the job.

   The spool file of a single step is emailed to the recipients.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Using Success and Failure Messages within an SAP Job Definition

You can check an SAP job's output for specific text strings to determine whether the job is a success or a failure. You specify the text string in the sap_success_msg or sap_fail_msg attributes in the job definition. For example, suppose when you cancel an SAP job you want it to complete to release its successor job. You can specify the text string 'Job canceled' as a success message in the job definition. When you cancel the job, the agent checks the job's spool file, finds a match for 'Job canceled' and marks the job as complete.

You can also check the output of a step (ABAP) to determine whether the step is a success or failure. You specify the text string in the sap_fail_msg=*value* and sap_success_msg=*value* keyword value pairs in the sap_step_parms attribute.

Note: The sap_step_parms: sap_fail_msg and sap_step_parms: sap_success_msg keywords do not apply to SAP Batch Input Session (SAPBDC) jobs.

For more flexibility, you can specify regular expressions instead of simple text strings within the success message and failure message fields. For example, you can use a regular expression to search for multiple strings at the same time. To compose a regular expression, follow the rules for Java class java.util.regex.Pattern. You can find these rules using a Google search for java pattern.

Note: To enable regular expression processing, you must configure the agent for the following parameter: sap.useRegularExpressions=true.

Examples: Using Regular Expressions

- This expression checks for "TEST" in the job output file. The first .* indicates that any number of characters can precede TEST. The second .* indicates that any number of characters can follow it.

  ```
  .*TEST.*
  ```

- This expression checks whether "not found" or "started" appears in the job output file.

  ```
  .*(not found|started).*
  ```

- This expression checks whether "Job canceled" appears in the job output file.

  ```
  .*Job\scanceled.*
  ```

Chapter 17: Secure Copy Jobs

This section contains the following topics:

Secure Copy Jobs

You can define a Secure Copy job to transfer binary files between an agent computer and a remote computer. The Secure Copy job can upload data to or download data from a remote server. The data is encrypted during the transfer. By default, a Secure Copy job uses the SFTP protocol. However, you can define the job to use the SCP protocol.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS. The agent must be configured as an FTP client using the Secure Copy Protocol or the Secure File Transfer Protocol.

Define a Secure Copy Job

You can define a Secure Copy (SCP) job to transfer binary files using the Secure Copy Protocol.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

To define a Secure Copy job

1. Insert a job and specify the following attributes in the definition:

   job_type: SCP

   Specifies that the job type is Secure Copy.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   scp_local_name

   Specifies a file on the agent computer to be downloaded or uploaded.

scp_remote_dir

Specifies the file's remote source directory (if downloading) or the file's remote destination directory (if uploading).

scp_remote_name

Specifies the file's source location (if downloading) or the file's destination (if uploading).

scp_server_name

Specifies a remote server name.

2. (Optional) Specify optional Secure Copy attributes:
   - job_class
   - scp_local_user
   - scp_protocol
   - scp_server_port
   - scp_target_os
   - scp_transfer_direction
3. (Optional) Specify common attributes that apply to all job types.

   The Secure Copy job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Upload a File Using the Secure File Transfer Protocol

This example uploads the logs.tar file to the /u/tmp directory on the hpsupport server. The job uses the Secure File Transfer Protocol (SFTP).

```
insert_job: sftp_upload
job_type: SCP
machine: WINAGENT
scp_transfer_direction: UPLOAD
scp_server_name: hpsupport
scp_remote_dir: /u/tmp
scp_remote_name: logs.tar
scp_local_name: "D:\temp\logs.tar"
scp_protocol: SFTP
owner: causer@WINAGENT
```

Note: The owner must be defined on CA Workload Automation AE using the autosys_secure command.

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following Secure Copy job attributes have default values:

scp_local_user

Specifies a user ID on the computer where the agent is installed. This user ID determines the access permissions on the agent computer.

Default: User that defined the job

scp_protocol

Specifies whether the SCP data transfer uses Secure File Transfer Protocol (SFTP) or regular Secure Copy (SCP).

Default: SFTP

scp_server_port

Specifies the port number of the remote server.

Default: 22

scp_target_os

Specifies the remote operating system type, which is used to determine the path separator on the remote system.

Default: UNIX

scp_transfer_direction

Specifies the file transfer direction between the agent computer and the remote server.

Default: DOWNLOAD

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Override the Transfer Direction

The scp_transfer_direction attribute in the following job definition overrides the default transfer direction.

This example uploads the logs.tar file to the /u/tmp directory on the hpsupport server. The job uses the Secure File Transfer Protocol (SFTP).

```
insert_job: sftp_upload
job_type: SCP
machine: WINAGENT
scp_transfer_direction: UPLOAD
scp_server_name: hpsupport
scp_remote_dir: /u/tmp
scp_remote_name: logs.tar
scp_local_name: "D:\temp\logs.tar"
scp_protocol: SFTP
owner: causer@WINAGENT
```

Chapter 18: Web Services Jobs

This section contains the following topics:

Web Service Jobs

The term web service describes a standardized method for exchanging data between applications and systems. Web services use XML to code and decode the data and Simple Object Access Protocol (SOAP) to transfer it.

Web Service Description Language (WSDL) is an XML-based language that describes a web service and how to access it. A WSDL document specifies the location of the service and the operations the service exposes.

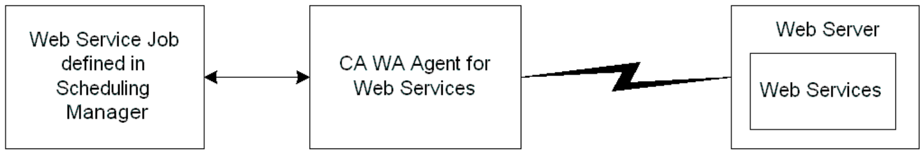
Universal Description, Discovery and Integration (UDDI) is an XML-based registry for businesses to list their available web services on the Internet. You can use the UDDI to access the WSDL.

Web services provide access to applications written in Java and Microsoft©.NET. A web service lets you invoke operations such as currency conversion, stock exchange quotes, or product pricing. In an enterprise workload automation environment, a web service might be used to invoke a business process such as posting accounts payable to the General Ledger. Some scheduling manager functions are also available as web services.

You can define a Web Service job to call an operation within a web service. The job passes parameters to the operation. The parameters can be actual values or a serialized Java object passed by another job. When the job invokes the web service, the parameters are passed to the operation. The job's output is stored by default as a serialized Java object in the job's spool directory. You can also specify a destination file for the output.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Web Services.

The following diagram shows the functional relationship between the scheduling manager, CA WA Agent for Web Services, and a web service residing on a web server:


Web Service Job defined in Scheduling Manager
CA WA Agent for Web Services
Web Server
Web Services

Note: If your company has a firewall and you must communicate through a proxy server to access a computer outside the firewall, agent configuration is required. For more information on configuring the agent for a proxy, see the *CA Workload Automation Agent for Web Services Implementation Guide*.

Define a Web Service Job

You can define a Web Service job to call an operation within a web service.

Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, or Windows and CA WA Agent for Web Services.

To define a Web Service job

1. Insert a job and specify the following attributes in the definition:

   job_type: WBSVC

   Specifies that the job type is Web Service.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   target_namespace

   Specifies the target namespace used for the names of messages, port type, binding, and services defined in the WSDL for the web service. Complex data types such as arrays require the target namespace.

   wsdl_operation

   Specifies the operation to be invoked.

2. (Optional) Specify optional Web Service attributes:
   - destination_file
   - endpoint_URL
   - job_class
   - one_way
   - port_name
   - return_class_name
   - return_namespace
   - return_xml_name
   - service_name
   - success_pattern
   - web_parameter
   - web_user
   - WSDL_URL

   Notes:

   - In a Web Service job, if you specify the WSDL_URL attribute but not the endpoint_URL attribute, you must specify both the service_name and port_name attributes. For the job to run successfully without the endpoint_URL attribute, the agent must be running on the same computer as the application server such as WebLogic or JBoss. If you specify both the WSDL_URL and endpoint_URL attributes, then the service_name and port_name attributes are optional.
   - The agent does not support document/literal styles of web services.
3. (Optional) Specify common attributes that apply to all job types.

   The Web Service job is defined. When the job runs, it calls an operation within a web service.

Notes:

- The one_way attribute is set to FALSE by default. If you do not specify this attribute in your job definition, the job waits for a response after the agent invokes the operation before completing. You can override this default setting by specifying the one_way attribute in your job definition.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Get a Company Stock Quote

Suppose that you want to invoke a web service that returns a company stock quote. The URL for the WSDL that describes the web service and its location is http://www.webservicex.com/stockquote.asmx?WSDL. The WSDL port name within the target namespace http://www.webserviceX.NET is StockQuoteSoap. The target endpoint address URL is http://www.webservicex.com/stockquote.asmx. The job calls the operation GetQuote within the StockQuote web service. When the job invokes the web service, the company's stock symbol is passed to the operation. The GetQuote operation returns a java.lang.String object, which maps to the XML type string in the return namespace http://www.webserviceX.NET/. When the job completes, the stock quote for CA is stored as a serialized Java object in the job's spool directory.

```
insert_job: quote
job_type: WBSVC
machine: wsagent
target_namespace: "http://www.webserviceX.NET/"
service_name: StockQuote
port_name: StockQuoteSoap
wsdl_operation: GetQuote
one_way: FALSE
WSDL_URL: "http://www.webservicex.com/stockquote.asmx?WSDL"
endpoint_URL: "http://www.webservicex.com/stockquote.asmx"
web_parameter: xsd\:string="CA"
return_class_name: java.lang.String
return_xml_name: string
return_namespace: "http://www.webserviceX.NET/"
```

Example: Validate an Email Address in a Web Service Job

Suppose that you want to invoke a web service that validates an email address. The URL for the WSDL that describes the web service and its location is http://www.webservicex.net/ValidateEmail.asmx?wsdl. The job calls the IsValidEmail operation within the ValidateEmail web service. When the job invokes the web service, the email address is passed to the operation. If the email address is valid, the operation returns true and the job completes successfully. If the email address is invalid, the operation returns false and the job fails.

```
insert_job: subscribe
job_type: WBSVC
machine: wsagent
target_namespace: "http://www.webserviceX.NET/"
service_name: ValidateEmail
port_name: ValidateEmailSoap
wsdl_operation: IsValidEmail
WSDL_URL: "http://www.webservicex.net/ValidateEmail.asmx?wsdl"
endpoint_URL: "http://www.webservicex.net/ValidateEmail.asmx"
web_parameter: xsd\:string="john.smith@example.com"
return_class_name: java.lang.Boolean
return_xml_name: boolean
return_namespace: "http://www.webservicex.net"
success_pattern: true
```

More information:

Insert a Job (see page 87)

Chapter 19: z/OS Jobs

This section contains the following topics:

z/OS Jobs

You can use z/OS jobs to run mainframe workload.

Note: To run these jobs, your system requires CA WA Agent for z/OS.

CA WA Agent for z/OS submits and tracks the z/OS jobs. You can define the following three types of z/OS jobs:

z/OS Regular

Schedules z/OS jobs.

z/OS Manual

Creates dependencies on z/OS jobs that are submitted outside of the scheduling manager.

z/OS Data Set Trigger

Creates dependencies on data set activities. You can customize trigger conditions to define the conditions in which the z/OS Data Set Trigger job completes. You can specify trigger conditions for the following data set activities:

- When a data set is created or updated
- When a specific job, group of jobs, or user ID creates a data set
- When an explicit data set notification is received (used when the data set activity does not generate an SMF record)
- When an FTP file is sent or received successfully

Note: Each data set must have its own individual z/OS Data Set Trigger job. To create dependencies on multiple data sets, you must create multiple z/OS Data Set Trigger jobs.

Define a z/OS Data Set Trigger Job

You can define a z/OS Data Set Trigger job to create dependencies on data set activities.

Note: To run these jobs, your system requires CA WA Agent for z/OS.

To define a z/OS Data Set Trigger job

1. Insert a job and specify the following attributes in the definition:

   job_type: ZOSDST

   Specifies that the job type is z/OS Data Set Trigger.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   zos_dataset

   Specifies the Job Control Language (JCL) library name. The JCL library or JCLLIB contains the JCL for the z/OS job. The JCLLIB is a z/OS data set name.

2. (Optional) Specify optional z/OS Data Set Trigger attributes:
   - zos_dsn_renamed
   - zos_dsn_updated
   - zos_explicit_dsn
   - zos_ftp_direction
   - zos_ftp_host
   - zos_ftp_userid
   - zos_trigger_by
   - zos_trigger_on
   - zos_trigger_type

3. (Optional) Specify common attributes that apply to all job types.

   The z/OS Data Set Trigger job is defined.

Notes:

- Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. If you specify the attribute, it overrides the default.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Define a z/OS Data Set Trigger Job

This example triggers when the data set PROD.CICS.FILE1602 is closed (created or updated).

```
insert_job: PROD.NIGHTLY
job_type: ZOSDST
machine: ZOS1
zos_dataset: PROD.CICS.FILE1602
owner: zosuser
```

More information:

Insert a Job (see page 87)

Attributes with Default Values

Attributes that have a default value automatically apply to the job definition. Therefore, you do not have to specify those attributes in the definition. Your agent administrator can define some default values on the agent in the agentparm.txt file.

If you specify the attribute in a job definition, it overrides the default.

The following z/OS Data Set Trigger job attributes have default values:

zos_explicit_dsn

Specifies whether the job monitors for an explicit data set.

Default: FALSE (The job does not monitor for an explicit data set.)

zos_dsn_renamed

Specifies whether the job monitors when a data set is renamed.

Default: N (The job does not monitor when the data set is renamed.)

zos_dsn_updated

Specifies whether the job monitors for updates to a data set.

Default: N (The job does not monitor for updates to the data set.)

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Monitor Data Set Activity by a User or Job

You can define a z/OS Data Set Trigger job to monitor when a specific job, group of jobs, or user ID creates a data set. When the specified condition is met, the job completes.

To monitor activity by a z/OS user or job

1. Define a z/OS Data Set Trigger job (see page 410).
2. Add the following attributes to the job definition:

   zos_trigger_type

   Specifies whether the job monitors data set activity by a job or a user ID.

   zos_trigger_by

   Specifies the name of the job or user who performs the data set activity that triggers the job.
3. Run the job.

   The job monitors for data set activity by the specified user or job.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Restrict the Trigger to Specific Data Sets Created by a Particular User

Suppose that you want the z/OS Data Set Trigger job PROD.PAY_DATA to release its successors when the user CYB1 creates generation data set USER1.PAYROLL (USER1.PAYROLL.G-). The agent ZOS1 monitors the data set under user CYBDL01.

```
insert_job: PROD.PAY_DATA
job_type: ZOSDST
machine: ZOS1
owner: CYBDL01
zos_dataset: USER1.PAYROLL.G-
zos_trigger_type: zos_user_id
zos_trigger_by: CYB1
```

Example: Restrict the Trigger to Specific Data Sets Created by a Particular Job

Suppose that you want a z/OS Data Set Trigger job named PROD.PAY_DATA to release its successors when job ABC creates generation data set USER1.PAYROLL (USER1.PAYROLL.G-).The agent ZOS1 monitors the data set under user CYBDL01.

```
insert_job: PROD.PAY_DATA
job_type: ZOSDST
machine: ZOS1
owner: CYBDL01
zos_dataset: USER1.PAYROLL.G-
zos_trigger_type: zos_job_name
zos_trigger_by: ABC
```

Monitor an FTP Transfer on z/OS

You can define a z/OS Data Set Trigger job to monitor when an FTP file is sent or received successfully. When the specified condition is met, the job completes.

To monitor for FTP transfers on z/OS

1. Define a z/OS Data Set Trigger job (see page 410).
2. Add the zos_explicit_dsn attribute to the job definition using the following syntax:

   ```
   zos_explicit_dsn: FALSE
   ```

3. Add the following attributes:

   zos_ftp_direction

   Specifies whether the job monitors for an FTP transfer to a remote computer or from a remote computer.

   zos_ftp_host

   Specifies the name of the remote computer involved in the FTP transfer. The data is transferred to or from the local mainframe computer.

   zos_ftp_userid

   Specifies the FTP user ID used to connect to a remote computer.

4. Run the job.

   The job monitors for the specified FTP transfer.

Note: For more information about attributes and their JIL syntax, see the *Reference Guide*.

Example: Monitor for a Data Set Sent to a Remote FTP partner

Suppose that you want the z/OS Data Set Trigger job CYBER.XFER to release its successors when data set CYBER.XFER.001 is successfully sent from the local mainframe partner to a remote FTP partner. The agent ZOS1 monitors the FTP transfer under user CYBDL01.

```
insert_job: CYBER.XFER
job_type: ZOSDST
machine: ZOS1
owner: CYBDL01
zos_dataset: CYBER.XFER.001
zos_ftp_direction: SEND
```

Example: Restrict Triggering to a Specific Host

Suppose that you want the z/OS Data Set Trigger job CYBER.XFER to release its successors when a remote FTP partner with IP address 172.16.0.0 successfully transfers a file creating the data set CYBER.XFER.001. The agent ZOS1 monitors the FTP transfer under user CYBDL01.

```
insert_job: CYBER.XFER
job_type: ZOSDST
machine: ZOS1
owner: CYBDL01
zos_dataset: CYBER.XFER.001
zos_ftp_direction: RECEIVE
zos_ftp_host: 172.16.0.0
zos_ftp_userid: CYB1
```

Example: Restrict Triggering to a Specific Logon ID

Suppose that you want the z/OS Data Set Trigger job CYBER.XFER to release its successors when a remote FTP partner successfully transfers a file creating the data set CYBER.XFER.001, assuming that the remote FTP partner logged on to the FTP server with the CYBER005 user ID. The agent ZOS1 monitors the FTP transfer under user CYBDL01.

```
insert_job: CYBER.XFER
job_type: ZOSDST
machine: ZOS1
owner: CYBDL01
zos_dataset: CYBER.XFER.001
zos_ftp_direction: RECEIVE
zos_ftp_host: 172.16.0.0
zos_ftp_userid: CYBER005
```

Example: Restrict the Trigger to an FTP Transfer from a Specific User ID

This example releases the job's successors when a remote FTP partner successfully transfers a file creating the data set CYBER.XFER.001, assuming that the user ID prefix of the local FTP partner is CYB (CYB-). The agent ZOS1 monitors the FTP transfer under user CYBDL01.

```
insert_job: CYBER.XFER
job_type: ZOSDST
machine: ZOS1
owner: CYBDL01
zos_dataset: CYBER.XFER.001
zos_trigger_type: zos_user_id
zos_trigger_by: CYB-
zos_ftp_direction: RECEIVE
zos_ftp_userid: CYB-
```

Define a z/OS Manual Job

You can define a z/OS Manual job to create dependencies on z/OS jobs that are submitted outside the scheduling manager, such as a job that is submitted manually by a user.

Note: To run these jobs, your system requires CA WA Agent for z/OS.

To define a z/OS Manual job

1. Insert a job and specify the following attributes in the definition:

   job_type: ZOSM

   Specifies that the job type is z/OS Manual.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   zos_jobname

   Specifies the name of the z/OS job that is submitted outside of CA Workload Automation AE.

2. (Optional) Specify optional z/OS Manual attributes:
   - auth_string
   - job_terminator
   - search_bw
3. (Optional) Specify common attributes that apply to all job types.

   The z/OS Manual job is defined.

Notes:

- The job_terminator attribute is set to N by default. If you do not specify this attribute in your job definition, the job does not terminate if its containing box completes with a FAILURE or TERMINATED status. You can override this default setting by specifying the job_terminator attribute in your job definition.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Post a z/OS Manual Job as Complete Based on the User ID

This example posts a z/OS Manual job as complete when the manually-submitted job ABC runs under user CYBER. The ZOS1 agent monitors job ABC.

```
insert_job: ABC_job
job_type: ZOSM
machine: ZOS1
zos_jobname: ABC
owner: zosuser
auth_string: CYBER
```

More information:

Insert a Job (see page 87)

Define a z/OS Regular Job

You can define a z/OS Regular job to schedule a z/OS job.

Note: To run these jobs, your system requires CA WA Agent for z/OS.

To define a z/OS Regular job

1. Insert a job and specify the following attributes in the definition:

   job_type: ZOS

   Specifies that the job type is z/OS Regular.

   machine

   Specifies the name of an existing machine definition where the agent runs.

   jcl_library

   Specifies the Job Control Language (JCL) library name. The JCL library or JCLLIB contains the JCL for the z/OS job.

2. Specify the following attribute if jcl_library is a partitioned data set (PDS):

   jcl_member

   Specifies the JCL member that contains the JCL for your job.

3. (Optional) Specify optional z/OS Regular attributes:

   - condition_code
   - copy_jcl
   - envvars
   - job_terminator

4. (Optional) Specify common attributes that apply to all job types.

   The z/OS Regular job is defined.

Notes:

- The job_terminator attribute is set to N by default. If you do not specify this attribute in your job definition, the job does not terminate if its containing box completes with a FAILURE or TERMINATED status. You can override this default setting by specifying the job_terminator attribute in your job definition.
- For more information about the optional attributes, common job attributes, and JIL syntax, see the *Reference Guide*.
- You can also use CA WCC to define the job. For more information about using CA WCC to define the job, see the *CA Workload Control Center Online Help*.

Example: Store a Working Copy of the JCL that You Submitted

Suppose that the agent ZOS1 submits the JCL in member CYBDL01A in the CYBDL01.JCLLIB library. If the job fails, you can modify a working copy of the JCL in the CYBDL01.COPY.JCLLIB data set, and resubmit the job without affecting the JCL source.

```
insert_job: CYBDL01A
job_type: ZOS
machine: ZOS1
jcl_library: CYBDL01.JCLLIB
jcl_member: CYBDL01A
owner: CYBDL01
copy_jcl: CYBDL01.COPY.JCLLIB
```

More information:

Insert a Job (see page 87)

Chapter 20: Working with User-defined Job Types

This section contains the following topics:

User-Defined Job Types

CA Workload Automation AE lets you define simple user-defined jobs. A user-defined job type is similar to a command job except that each user-defined job type is associated with a custom program/script/adaptor. For example, a new job type can be defined to perform FTP. For this to work, a custom program/script/adaptor has to be provided which does FTP by taking a few arguments. Jobs using the user-defined job type may optionally specify arguments to the command defined in the user-defined job type. For example, if we define FTP job type as '2', a custom program, script, or adaptor must be provided as part of the definition of type 2.

```
insert_job_type: 2
command: /home/scripts/myftp
```

When jobs of type 2 are defined, all of them execute /home/scripts/myftp when those jobs are run.

```
insert_job: ftp_test
job_type: 2
machine: localhost
std_in_file: /tmp/ftp_params
```

When a new version of FTP script is used, only the definition of job type has to be modified.

You can use the following jil commands to create, update, and delete user-defined job types.

- insert_job_type
- delete_job_type
- update_job_type

Only three attributes are associated with user-defined job types:

job_type

Defines the user-specified job type.

Limits: This value can be a singe-digit number (0-9).

command

Defines the command to associate with the job type.

Limits: This value can be up to 510 characters in length.

The command attribute in the job definition is optional. If you do not specify the command attribute in the job definition, the job type uses the command attribute of the job definition. If you specify the command attribute in the job definition, it appends the command attribute to the job type command.

description

Defines a description of the job type.

Limits: This value can be up to 256 characters in length.

Any other attribute is rejected and JIL fails.

Note: You must define a job type before you can use it to define a job. For more information, see the *Reference Guide.*

Example: Use insert_job_type to Add a User-Defined Job Type

This example creates an association between a user-defined job type and an executable.

```
insert_job_type: 5
description: Web Service Adapter
command:ws.exe
```

Example: Use delete_job_type to Delete a User-Defined Job Type

This example verifies that no jobs are currently using the specified job type, and deletes the job type.

```
delete_job_type: 5
```

Example: Use update_job_type to Modify a User-Defined Job Type

This example modifies an existing job type, changing the values of the description and command attributes.

```
update_job_type: 5
description: WorldView Adapter
command: wv.exe
```

Example: Pass Arguments To a User-Defined Job Type Command

This example creates a new job type. Two jobs are defined that run with different arguments. The user-defined job is defined to run the command /bin/sleep. Each job specifies the sleep time as an arugment that is appended to the command. For example, the job sleep_5 would be executed with a command /bin/sleep 5.

```
update_job_type: 7
description: sleep for a spell
command: /bin/sleep

insert_job: sleep_5
description: sleep for 5 seconds
job_type: 7
machine: localhost
command: 5

insert_job: sleep_30
description: sleep for 30 seconds
job_type: 7
machine: localhost
command: 30
```

Create a New Job Type

You can create new job types. For example, you have an adapter binary and you want to create 300 jobs that invoke the adapter. You can create 300 command jobs and specify the command each time or you can define a single job type (for example '0') that represents the adapter command and define 300 jobs of type '0'.

To create a new job type

1. Insert_job_type:0.
2. Enter the following command: special_adapter.
3. Enter the following description: This is a job type to run special adapter commands.

   The new job type is created.

Use a New Job Type

After you create a new job type, you must define a job to use it.

To use a new job type

1. Insert_job:test.
2. Enter job_type:0.
3. Enter machine name: localhost.

   The newly created job type can be used.

CA Workload Automation AE also supports delete_job_type and update_job_type. You can use the delete_job_type to delete a user-defined job type, and the update_job_type when you want to modify an existing user-defined job type.

Chapter 21: Working with Resources

This section contains the following topics:

Real Resources

Real resources are system conditions that are directly tied to a physical system (for example, physical memory). Real resources are predefined to CA Workload Automation AE and are managed by external resource managers such as CA Spectrum Automation Manager (formerly named CA DCA Manager). You cannot define or update real resources using CA Workload Automation AE. If the required resources are not available, the job goes into a RESWAIT state and is not submitted until the resources are available.

You can specify real resources as dependencies to jobs. A job with resource dependencies is submitted only when the resources required are available. If the required resources are not available, the job goes into a RESWAIT state and is not submitted until the resources are available.

You can specify the following supported real resource types as dependencies:

CPU_IDLE_PCT

Defines the percentage of time over the sample period that the system's CPUs were idle.

Example: (CPU_IDLE_PCT, VALUE=50, VALUEOP=GT) indicates that the system's CPUs were idle for at least 50% of the sample period.

Corresponding metric in the CA SystemEDGE agent: cpuTotalIdlePercent

CPU_LOAD_AVG_15MIN

Defines the load average in the last 15 minutes.

Example: (CPU_LOAD_AVG_15MIN, VALUE=50, VALUEOP=LT) indicates that the load average was less than 50 in the last 15 minutes.

Corresponding metric in the CA SystemEDGE agent: loadAverage15Min

CPU_LOAD_AVG_5MIN

Defines the load average in the last 5 minutes.

Example: (CPU_LOAD_AVG_5MIN, VALUE=50, VALUEOP=LT) indicates that the load average was less than 50 in hte last 5 minutes.

Corresponding metric in the CA SystemEDGE agent: loadAverage5Min

MEM_INUSE_PCT

Defines the percentage of the system's active memory that is in use.

Example: (MEM_INUSE_PCT, VALUE=30, VALUEOP=LTE) indicates that 30% or less of the system's active memory is in use.

Corresponding metric in the CA SystemEDGE agent: memCapacity

SWAP_INUSE_PCT

Defines the percentage of the system's total swap that is in use.

Example: (SWAP_INUSE_PCT, VALUE=60, VALUEOP=LT) indicates that less than 60% of the system's total swap is in use.

Corresponding metric in the CA SystemEDGE agent: swapCapacity

SWAP_SPACE_TOTAL

Defines the total swap space (in KB).

Example: (SWAP_SPACE_TOTAL, VALUE=10240, VALUEOP=GTE) indicates that the total swap space is 10240 KB or more.

Corresponding metric in the CA SystemEDGE agent: totalSwapSpace

SYSTEM_CPU_COUNT

Defines the total number of CPUs that the job requires.

Example: (SYSTEM_CPU_COUNT, VALUEOP=EQ, VALUE=2) indicates that the job requires a machine with 2 CPUs.

Corresponding metric in the CA SystemEDGE agent: Number of CPUs

SYSTEM_CPU_SPEED

Defines the system clock speed (in MHz) the job requires.

Example: (SYSTEM_ CPU_SPEED, VALUEOP=EQ, VALUE=100) indicates that the job requires a machine with clock speed of 100MHz.

Corresponding metric in the CA ACM agent: CPU Speed

SYSTEM_OS_TYPE and VERSION

Specifies the operating system name and version number that the job requires.

Example: (SYSTEM_OS_TYPE, VALUEOP=EQ, VALUE=AIX VERSION=5.3) indicates that the job requires a machine with an AIX 5.3 operating system.

Corresponding metrics in the CA SystemEDGE agent: OS Type/OS Version

SYSTEM_PHYSICAL_MEMORY

Defines the total amount of available physical memory (in MB) that the job requires.

Example: (SYSTEM_PHYSICAL_MEMORY, VALUEOP=GTE, VALUE=200) indicates that the job requires a machine with at least 200 MB of physical memory.

Corresponding metric in the CA SystemEDGE agent: Physical Memory

SOFTWARE_NAME and VERSION

Specifies the software and version number that the job requires.

Example: (SYSTEM_SOFTWARE_NAME, VALUEOP=EQ, VALUE=Adaptive Enterprise Server Sybase, VERSION=15.0) indicates that the job requires a machine that has Adaptive Enterprise Server Sybase 15.0 installed.

Corresponding metrics in the CA ACM agent: SOFTWARE_NAME/VERSION

The previous examples show the syntax for specifying the resource value in a real resource dependency definition.

Notes:

- To use real resource dependencies, you must install and configure the CA Spectrum Automation Manager SDK clients on the CA Workload Automation AE scheduler and application server machines. You must also install the CA SystemEDGE agent on the machines where the real resources are located. You need to also install CA ACM agents if you want to use SOFTWARE and SYSTEM_CPU_SPEED metrics.
- For more information about configuring CA Workload Automation AE to work with CA Spectrum Automation Manager, see the *UNIX Implementation Guide* or *Windows Implementation Guide*.

Virtual Resources

Virtual resources can help you control job execution and improve your environment's performance. They represent values that can be quantified, but they are not directly tied to a physical system. CA Workload Automation AE does not check whether a virtual resource is an actual device that exists or whether it is a device that is being used by another process.

You can use virtual resources to prevent jobs from running simultaneously and ensure that a job is submitted only when the minimum number of resources is available. For example, you can define a virtual resource to represent the maximum number of floating product licenses available in your enterprise. Each time a qualified job runs, a unit of that resource is used. When all the units are used, no more jobs can run.

You can define the following types of virtual resources on CA Workload Automation AE:

- Depletable
- Renewable
- Threshold

A virtual resource can be defined for a specific machine, or it can be defined at the global level. A resource defined for a specific machine is available to jobs submitted to that machine. A resource may be defined to more than one specific machine. A resource at the global level is available to all machines controlled by CA Workload Automation AE. Global resources can help control workload balancing across all machines.

Note: You can define resources for distributed machines only. Before you can define a resource on a machine, the machine must already be defined on the database.

You can specify virtual resources as dependencies to jobs. A job with resource dependencies is submitted only when the resources required are available. If the required resources are not available, the job goes into a RESWAIT state and is not submitted until the resources are available.

Virtual resources are associated with corresponding resource pools. When jobs with virtual resource dependencies run, the used resources are temporarily or permanently removed from the resource pool, depending on the virtual resource type.

CA Workload Automation AE is the resource manager for virtual resources.

Note: Virtual resources are secured using CA EEM (external security mode). The default native security in CA Workload Automation AE does not secure virtual resources. For more information about securing virtual resources using CA EEM, see the *Security Guide*.

Depletable Resources

A depletable resource is a consumed resource. When a job that uses this resource is submitted, the used resource units are permanently removed from the resource pool. When the resource is completely depleted or jobs require more units than what is currently available, jobs that need it go into a RESWAIT state and are not submitted until the resource is available. You can manually replenish the resource using the update_resource JIL subcommand. After the resource is replenished, other jobs can use it.

Depletable resources are helpful when you want to represent values that have a limit, such as the maximum times to run a job. Depletable resources are also helpful when you want to control how many times a job can run in a specified time period.

Example: Run a Job Only Once a Day

Suppose that a bank wants to post daily transactions to its master database only once a day at midnight to ensure data integrity and system performance. To run this critical job only once a day, you can do the following:

1. Define a depletable resource with an amount of 1, as follows:

   ```
   insert_resource: depletable1
   res_type: D
   machine: hostname
   amount: 1
   ```

2. Define the critical job with the following conditions:

   - The job requires one unit of the depletable resource before it can start.
   - The job is scheduled to start at midnight.

   The job is defined as follows:

   ```
   insert_job: ResDepJob
   job_type: CMD
   command: /tmp/DBIntensiveApp
   machine: hostname
   owner: root@hostname
   resources: (depletable1,QUANTITY=1)
   ```

When the job is submitted, one unit of the resource is permanently removed from the resource pool. If the job fails, CA Workload Automation AE restarts the job (based on the n_retrys attribute), but the job goes into RESWAIT state because no units of the resource are available. The RESWAIT state indicates that a potential problem occurred. You must run the job must manually.

Note: To add additional resources to a depletable resource, use the update_resource subcommand. For example, the following command adds 15 units to the depletable1 resource:

```
update_resource: depletable1
machine: hostname
amount: 15
```

Renewable Resources

A renewable resource is a borrowed resource. When a job that uses this resource is submitted, the used resource units are temporarily removed from the resource pool. When the job completes, the resource units are returned to the pool, or the units are held until they are manually released back to the pool. Renewable resources are helpful when you want to control jobs that run concurrently or serially.

When the resource is being used, other jobs that need more units than what is currently available go into a RESWAIT state and are not submitted until the resource is available. You can change the amount of resource units available using the update_resource JIL subcommand. After the amount is changed, a greater number of jobs that need the resource can run concurrently.

Example: Control the Maximum Number of Licenses Used

Suppose that your enterprise has 10 floating licenses for a program. Multiple licenses can be used on one machine, or they can be used on up to 10 machines. At any time, you want to ensure that the maximum number of licenses used is 10. To control the maximum number of licenses being used, you can do the following:

1. Define a renewable resource at the global level with an amount of 10, as follows:

   ```
   insert_resource: r1
   res_type: R
   amount: 10
   ```

2. Define each job that requires the license with the following conditions:

   - The job requires one unit of the renewable resource before it can start.
   - The job frees the renewable resource whether it completes successfully or not.

   The job is defined as follows:

   ```
   insert_job: jr1
   command: sleep 500
   machine: hostname
   resources: (r1,quantity=1, FREE=A)
   ```

When a job is submitted, one unit of the resource is temporarily removed from the resource pool. Because there are only 10 units, only 10 of these jobs can run simultaneously on any machine in the enterprise. Other jobs that require a license cannot be submitted because no resources are available. When a job that is running completes, one unit of the resource is returned to the resource pool. Another job can be submitted because a unit is now available.

Threshold Resources

A threshold resource is a sizing resource. For example, if the threshold resource is set to 2, CA Workload Automation AE submits the jobs that require 2 or fewer units. Threshold resources are helpful when you want to define a boundary that controls which jobs are submitted to run. The used resource units are not removed from the resource pool.

A job that has a dependency on a threshold resource is only submitted if it requires the available resource units or fewer. Otherwise, the job goes into a RESWAIT state and is not submitted until the threshold amount is increased. You can change the threshold amount using the update_resource JIL subcommand. After the amount is changed, the jobs that meet the threshold can run.

Example: Prevent Jobs from Running When a Critical Resource is Offline

Suppose that multiple jobs read and write to a disk on a machine named mach1. When the disk needs to be formatted, you want the jobs to stop running. To prevent the jobs from running when the disk is offline, you can do the following:

1. Define a threshold resource on the mach1 machine with an amount of 1, as follows:

   ```
   insert_resource: t1
   machine: mach1
   res_type: T
   amount: 1
   ```

2. Define each job that reads and writes to the disk and requires one unit of the threshold resource before it can start, as follows:

   ```
   insert_job: jt1
   command: readnadwritetodisk
   machine: mach1
   resources: (t1, quantity=1)
   ```

3. Define a job named TRIGGER_JOB that resets the threshold amount to 0 to indicate that the disk is offline. The trigger job runs the update_resource command as follows:

   ```
   update_resource: t1
   machine: mach1
   amount: 0
   ```

When the disk is online, the threshold resource amount is 1, so all jobs that need to read and write to the disk are submitted (the jobs meet the threshold requirement). When the disk needs to be formatted, you can run TRIGGER_JOB, which resets the threshold resource amount to 0. All the jobs that need to read and write to the disk go into a RESWAIT state and are not submitted until the threshold is set to 1.

Define a Virtual Resource

To use virtual resource dependencies in your jobs, you must first add the virtual resource definition to the database. You can define virtual resources on distributed machines only.

To define a virtual resource

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.
2. Specify the following subcommand and attributes:

   insert_resource: *resource_name*

   Specifies the virtual resource to be defined.

   amount

   Defines the number of units to assign to the virtual resource.

   res_type

   Specifies the virtual resource type (D for depletable, R for renewable, or T for threshold).
3. Specify the following additional attribute if you want to define the resource for a machine:

   machine

   Specifies the name of the machine to define the virtual resource for. The machine must already be defined on the database, and the type of the machine cannot be v, w, or p.

   Note: If you do not specify the machine attribute, the resource will be available to all machines (global level resource).
4. (Optional) Specify the following additional attribute:

   description

   Defines a free-form text description of the virtual resource.
5. Do *one* of the following:
   - Enter **exit** if you are using interactive mode.
   - Redirect the script to the jil command if you are using a script.

   The insert_resource subcommand is issued and the specified virtual resource is defined.

Notes:

- The virtual resource name must be unique across all resource types.
- You cannot define the same virtual resource at the machine level and global level.
- You can define the same virtual resource on multiple machines.
- For more information about the syntax for the insert_resource subcommand and related attributes, see the *Reference Guide*.

Example: Define a Global Depletable Resource

This example defines a virtual depletable resource named glob_res. The machine attribute is not specified, so the resource is available to all machines.

```
insert_resource: glob_res
res_type: D
amount: 50
description: "This resource is permanently consumed."
```

Example: Define a Machine-Level Threshold Resource

This example defines a threshold resource for the unixagent machine. The resource is assigned 10 units, so only jobs that require 10 or fewer units of this resource are submitted to the unixagent machine.

```
insert_resource: threshold_res
res_type: T
machine: unixagent
amount: 10
```

Update a Virtual Resource

You can update the amount and description properties of a virtual resource definition in the database. Updating the resource amount is helpful when you want to change the number of jobs that can run concurrently or serially.

To update a virtual resource

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.
2. Specify the following subcommand:

   update_resource: *resource_name*

   Specifies the virtual resource to be updated. This resource must be defined in the database.
3. Specify the following additional attribute if the resource is defined for a machine:

   machine

   Identifies the name of the machine that the virtual resource is defined for.

   Note: If you do not specify the machine attribute, CA Workload Automation AE assumes the resource is a global resource. If the resource is machine-level, but the machine attribute is not specified, you will get an error.
4. (Optional) Specify the following additional attributes:

   amount

   Defines the number of units to assign to the virtual resource. The number can be an absolute value or a relative value.

   description

   Defines a free-form text description of the virtual resource.
5. Do *one* of the following:
   - Enter **exit** if you are using interactive mode.
   - Redirect the script to the jil command if you are using a script.

   The update_resource subcommand is issued and the specified virtual resource is updated.

Notes:

- You cannot update resource types or machine names using the update_resource subcommand. To update resource types or machine names, you must delete the resource and add it to database again with the new properties.
- For more information about the syntax for the update_resource subcommand and related attributes, see the *Reference Guide*.

Example: Update a Machine-Level Virtual Resource

This example updates a virtual resource that is associated with the unixagent machine. The number of units is changed to 35.

```
update_resource: mach_res
machine: unixagent
amount: 35
```

Suppose that the amount attribute is defined as follows:

```
amount: +20
```

In this situation, the command adds 20 units to the available resource count. For example, if the resource already has 35 units, the command adds 20 units and the total would be 55.

Similarly, suppose that the amount attribute is defined as follows:

```
amount: -20
```

If the resource has 35 units, the command removes 20 units and the total would be 15.

Delete a Virtual Resource

You can delete a virtual resource that you no longer use.

Note: You cannot delete a virtual resource if it is referenced as a dependency in a job. To delete the resource, you must delete all the related job dependencies first. If a resource is deleted while a job that references it as a dependency is active or running, the job continues to run and is not affected.

To delete a virtual resource

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.
2. Specify the following subcommand:

   delete_resource: *resource_name*

   Specifies the virtual resource that you want to delete.
3. Specify the following additional attribute if the resource is defined for a machine:

   machine

   Identifies the name of the machine that the virtual resource is defined for.

   Note: If you do not specify the machine attribute, CA Workload Automation AE assumes the resource is a global resource. If the resource is machine-level, but the machine attribute is not specified, you will get an error.
4. Do *one* of the following:
   - Enter **exit** if you are using interactive mode.
   - Redirect the script to the jil command if you are using a script.

   The delete_resource subcommand is issued and the specified virtual resource is deleted.

Note: For more information about the syntax for the delete_resource subcommand and related attributes, see the *Reference Guide*.

Example: Delete a Global Virtual Resource

This example deletes the global virtual resource named glob_resource.

```
delete_resource: glob_resource
```

Define Real and Virtual Resource Dependencies in a Job

You can define a job to have dependencies on real and virtual resources. A job with resource dependencies is submitted only when the resources required are available. Using resource dependencies can help you control job execution and improve your environment's performance.

Note: You cannot define a resource dependency on a box.

To define real and virtual resource dependencies in a job

1. Insert a job and add the following attribute to the job definition:

   resources

   Defines one or more real and virtual resource dependencies.

   Note: To define a virtual resource dependency, the virtual resource must already be defined on the database.

2. (Optional) Add the following attribute:

   priority

   Defines the queue priority of the job. The queue priority establishes the relative priority of all jobs queued for a given machine. A lower number indicates a higher priority. Jobs with a higher priority get the required resources first and run before lower priority jobs.

3. Run the job.

   The job is defined with the specified resource dependencies. The job is submitted when the required resources are available.

Note: For detailed information about the syntax for the resources and priority attributes, see the *Reference Guide*.

Example: Define a Virtual Resource Dependency that Does Not Free the Resources After Job Completion

This example defines a Command job that has a dependency on a virtual renewable resource. Before the job can start running, it needs all the units of the renew_res resource.

```
insert_job: no_free_job
job_type: CMD
machine: unixagent
command: /u1/procrun.sh
resources: (renew_res, QUANTITY=ALL, free=N)
```

After the job completes, the units of the renew_res resource are not freed from the job. To return the units back to the available resource pool, you must issue the following command:

```
sendevent -E RELEASE_RESOURCE -J no_free_job
```

Suppose that the job is defined with the following attribute:

```
resources: (renew_res, QUANTITY=5, free=Y)
```

If the job completes successfully, the resource is added back to the resource pool. This is the default behavior.

Suppose that the job is defined with the following attribute:

```
resources: (renew_res, QUANTITY=5, free=A)
```

The resource is released back to the pool unconditionally.

Example: Define Real and Virtual Resource Dependencies

This example defines a Command job that has real and virtual resource dependencies. Before the job can start running, it needs a machine that satisfies all the following dependencies:

- 1 unit of the depletable resource named D1
- 3 units of the threshold resource named T1
- 4 units of the renewable resource named R1
- SYSTEM_OS_TYPE is AIX 5.3
- SYSTEM_PHYSICAL_MEMORY is greater than 2 GB

```
insert_job: res_dep_job
job_type: CMD
machine: unixagent
command: /u1/procrun.sh
resources: (D1, QUANTITY=1) AND (T1, QUANTITY=3) AND
(R1, QUANTITY=4, FREE=Y) AND
(SYSTEM_OS_TYPE, VALUEOP=EQ, VALUE=AIX, VERSION=5.3) AND
(SYSTEM_PHYSICAL_MEMORY, VALUEOP=GT, VALUE=2097152)
```

Update Real and Virtual Resource Dependencies in a Job

You can update the real and virtual resource dependencies defined in a job.

To update real and virtual resource dependencies in a job

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.

2. Specify the following subcommand and attribute:

   update_job: *job_name*

   Specifies the job to update.

   resources

   Specifies one or more real and virtual resource dependencies to update.

3. Do *one* of the following:

   - Enter **exit** if you are using interactive mode.
   - Redirect the script to the jil command if you are using a script.

   The update_job subcommand is issued and the specified real and virtual resource dependencies are updated.

Note: For detailed information about the syntax for the resources attribute, see the *Reference Guide*.

Example: Update Real and Virtual Resource Dependencies

This example updates the real and virtual resources in the proc_daily job. RENEW3 virtual resource dependency does not have the free keyword defined, so the resource units are freed if the job completes successfully (the default).

```
update_job: proc_daily
resources: (RENEW3, QUANTITY=3) AND (THRESHOLD2, QUANTITY=3) AND (SYSTEM_CPU_COUNT,
VALUEOP=GT, VALUE=2)
```

Release Renewable Resources

When you define a job to have a virtual renewable resource dependency, you can specify whether the units of the resource are freed after the job completes. If the units are not freed, you can manually release them back to the resource pool so that other jobs can use them.

To manually release renewable resources, enter the following command at the UNIX operating system prompt or the Windows instance command prompt:

```
sendevent -E RELEASE_RESOURCE -J job_name
```

job_name

Specifies the job that you want to release the renewable resources from.

The sendevent command is issued and the resources are released from the job.

Note: For more information about the sendevent command, see the *Reference Guide*.

Example: Release Renewable Resources

Suppose that you defined a virtual renewable resource named ren1 that has two units. JobA and JobB are also defined with a dependency on ren1. Each job requires one unit of ren1 before it can run. The jobs release the resource units only if they complete successfully.

When the jobs are submitted, the required units are temporarily removed from the resource pool. Suppose that JobA completes successfully and JobB fails. JobA returns one unit of resource to the resource pool. The other unit used by JobB is held, so the resource pool only has one unit. Other jobs that require more than one unit go into a RESWAIT state until the required units are available.

The following command manually releases the one unit of resource held by JobB so that other jobs can use it:

```
sendevent -E RELEASE_RESOURCE -J JobB
```

Generate a Report on Current Resource Definitions

You can generate a report that displays the current resource definitions in a database. Generating a report is helpful when you want to determine the properties of a resource and the values that can be specified when defining resource dependencies in jobs.

To generate a report on the current resource definitions, enter the following command at the UNIX operating system prompt or the Windows instance command prompt:

```
autorep –V resource_name -q
```

resource_name

Specifies the virtual resource definition that you want to generate a report on. You can use wildcard characters.

The autorep command is issued and the report is generated.

Note: For more information about the autorep command, see the *Reference Guide*.

Generate a Report to Display a Job's Resource Dependencies

You can generate a report that displays a job's current definition, including its resource dependencies.

To generate a report to display a job's resource dependencies, enter the following command at the UNIX operating system prompt or the Windows instance command prompt:

```
autorep –J job_name -q
```

job_name

Specifies the job that you want to generate a report for.

The autorep command is issued and the report is generated. The job's resource dependencies are included in the report.

Note: For more information about the autorep command, see the *Reference Guide*.

Generate a Report to Monitor Virtual Resource Usage

You can generate a report that monitors virtual resource usage. This report is helpful when you want to check how many units a virtual resource has defined, how many units are currently available, and which machines are using the resource during run time.

Note: The detail report (-d option) only displays the jobs that are *currently* using the resource.

To generate a report to monitor virtual resource usage, enter *one* of the following commands at the UNIX operating system prompt or the Windows instance command prompt:

- To generate a summary report:

  ```
  autorep –V resource_name -s
  ```

- To generate a summary report and detail report:

  ```
  autorep –V resource_name -d [-M machine_name]
  ```

-V *resource_name*

Specifies the virtual resource definition that you want to generate a report for. You can use wildcard characters.

Note: You can specify ALL to generate a report on all resource definitions.

-s

Generates a summary report.

-d

Generates a summary report and detail report.

[-M *machine_name*]

(Optional) Filters the detail report to display only the specified machine. You can use wildcard characters to match machine names.

Default: ALL (If you do not specify the -M option, the report displays all machines that are using the resource.)

The autorep command is issued and the report is generated.

Note: For more information about the autorep command, see the *Reference Guide*.

Example: Generate a Summary Report for a Global Virtual Resource

This example generates a summary report for the virtual resource named ren1. The resource is global, so it is available to all machines.

```
autorep -V ren1 -s
```

The report displays the following information:

```
Name    Machine   Defined   Available
----    -------   -------   ---------
ren1      --         4      1
```

The resource has a total of 4 units defined. It has 1 unit available, so 3 units are currently being used by jobs.

Example: Generate Summary and Detail Reports for a Global Virtual Resource

This example generates a summary report and a detail report for the ren1 virtual resource. The resource is global, so it is available to all machines.

```
autorep -V ren1 -d
```

The resulting report might resemble the following:

```
Resource Name              Type       Machine              Defined Available

__________________________ __________ ____________________ _______ _________
ren1                       V          ---                  4       1

/**** Current Resource Usage ****/

ResName            JobName            Run/Ntry    Status        Machine     Amount in Use

__________________ __________________ ___________ _____________ ___________ _____________
ren1               job1               1/1         RUNNING       machine1    1
ren1               job2               1/1         RUNNING       machine2    2
```

The -M option is not specified in the command, so the report displays all the jobs that are currently using the resource.

Example: Generate Summary and Detail Resource Reports for Individual Machines

Suppose that the virtual renewable resource named res_count is defined on three different machines (amachine1, machine2, and bmachine3). This example generates a summary report and a detail report for the res_count resource.

```
autorep -V res_count -d -M amachine*
```

The report displays the following information:

```
Name        Machine    Defined   Available
----        -------    -------   ---------
res_count   amachine1     4           0
res_count   amachine2     3           1

/**** Current Resource Usage ****/

ResName     JobName    Run/Ntry    Status    Machine    Amount in Use
-------     -------    --------   ------    -------    -------------
res_count    Job1      1/1              RUNNING   amachine1      4
res_count    Job2      2/1              RUNNING   amachine2      2
```

The report displays the jobs that are currently running. The -M option is specified with a wildcard, so the report only displays the jobs on the machines whose names match the -M value.

Generate a Report to Monitor Resource Dependencies

You can generate a report that monitors virtual resource dependencies. This report is helpful when you want to check which resources a job depends on and which of the resource dependencies are satisfied.

To generate a report to monitor virtual resource dependencies, enter the following commands at the UNIX operating system prompt or the Windows instance command prompt:

```
job_depends -J job_name -r
```

job_name

Specifies the job that you want to generate a dependency report for.

The job_depends command is issued and the report is generated.

Note: For more information about the job_depends command, see the *Reference Guide*.

Example: Generate a Report on Virtual Resource Dependencies

Suppose that you define three virtual resources (dep1, ren1, and thr1). The resources are global, so they are available to all machines.

```
insert_resource: dep1
res_type: D
amount: 0

insert_resource: ren1
res_type: R
amount: 0

insert_resource: thr1
res_type: T
amount: 1
```

You also define a job named job1 that belongs to a machine group that includes the M1 and M2 machines. job1 depends on all three virtual resources.

```
insert_job: job1
job_type: CMD
command: &sleep 500
machine: machineGroup1      /* This machine group includes machines M1 and M2. */
resources: (ren1, QUANTITY=1, FREE=A) AND (dep1, QUANTITY= 1) AND (thr1, QUANTITY=
1)
```

The following command generates a report on job1's resource dependencies on the M1 and M2 machines:

```
job_depends -J job1 -r
```

The report displays the following information:

```
Job Name        Machine
--------        ----------
job1                   M1

        Virtual Resources
        -----------------

ResourceType    Amount  Satisfied?
--------      ----      ----    ----------
ren1            R       1       NO
dep1            D       1       NO
thr1            T       1       YES
```

```
Job Name        Machine
--------        -------
job1                   M2

        Virtual Resources
        -----------------

ResourceType    Amount  Satisfied?
------------    ----    ----------
ren1            R       1       NO
dep1            D       1       NO
thr1            T       1       YES
```

Example: Generate a Report on Virtual and Real Resource Dependencies

Suppose that you define a job with virtual and real resource dependencies as follows:

```
insert_job: jobA
job_type: CMD
command: &sleep 500
machine: M1  /* This is an AIX machine */
resources: (ren1, QUANTITY=1, FREE=A) AND (thr1, QUANTITY= 1) AND (SYSTEM_OS_TYPE,
VALUEOP=EQ, VALUE=AIX)
```

The following command generates a report on job1's resource dependencies on the M1 machine:

```
job_depends -J job1 -r
```

The report displays the following information:

```
Job Name        Machine
--------        ----------
jobA                   M1
        Virtual Resources
        -----------------
ResourceType    Amount  Satisfied?
------------    ------  ----------
ren1            R       1       NO
thr1            T       1       YES

Real Resources
---------------------
Resource                                Satisfied?
-----------                                     -----------
SYSTEM_OS_TYPE, VALUEOP=EQ, VALUE=AIX   YES
```

Note: If the SysEdge agent is not available (that is, it is not installed or not running) on the machine M1, all the real resources dependencies are disqualified although they satisfy the resource dependency criterion. In this example, if the SysEdge agent is not running, the Satisfied field displays "NO”.

Chapter 22: Working with Binary Large Objects (Blobs)

This section contains the following topics:

Binary Large Objects

Binary Large Objects (blobs) are binary data of variable length. CA Workload Automation AE supports blobs in job definitions, and after they are defined, they are stored in the database. This allows the blob data to be shared by jobs running on multiple computers.

To understand the advantages of using blobs in CA Workload Automation AE environment, refer to the following example, which explains the process that is used to share data amongst the jobs that are running on a single computer:

1. When the jobs are running on a single computer, you can define a command job to run a program that outputs the data to a file using the std_out_file attribute.
2. When the job is completed, a file is created in the location specified by the std_out_file attribute.
3. All the other jobs that depend on this output data can access this file.
4. You can also define a second command job to run a program that reads the output data of the previous job, by specifying the file name in the std_in_file attribute.
5. This second command job opens the file specified by the std_in_file attribute and passes the data to the program, allowing it to complete successfully.

Based on this example, as the output data is stored in a file on one computer, it is not available to all the other jobs that are scheduled to run on other computers. However, the use of blobs allows the data that is saved as output by a job on one computer to be shared by all the other jobs that are running across multiple computers.

Also, you can define a command job to run a program that uploads the output data to the database as a blob using the std_out_file attribute. You can also define a second command job to run a program that reads the blob data of the previous job using the std_in_file attribute. The second command job downloads the blob data specified by the std_in_file attribute from the database and passes the data to the program, allowing it to complete successfully.

Blob data can be of the following types:

Binary Data

Requires a program that understands the format of the data to interpret the bytes in binary data. For example:

Multimedia files which include the following:

- Images
- Video files
- Audio files

Textual Data

Requires an operating system that can interpret the bytes in textual data, which contains the characters that conform to the ASCII standard.

Note: Some operating systems handle the specification of a new line in the textual data differently. In this instance, you must convert the necessary textual data when it is copied across operating systems.

CA Workload Automation AE allows you to specify the type of blob data that is being used and converts the textual data when it is downloaded across multiple operating systems.

Types of Blobs

CA Workload Automation AE supports the following types of blobs:

- Job blobs
- Global blobs

Job Blobs

Job blobs are associated with an existing CA Workload Automation AE job and are referenced by the job name. Job blobs can either be created at the time of the job definition or after the job has been defined. They are deleted when the job is deleted.

There are three types of job blobs, which include the following:

Input

Contains the input data that is reserved for the job to which they are associated in textual data format.

Output

Stores the program output messages of a running job in textual or binary data format.

Error

Stores the error messages of a running job in textual or binary data format.

Input Job Blobs

Input blobs are uploaded to the database using JIL. You can insert an input job blob multiple times. Each time it is inserted, it acquires a new version number.

When the job starts, the most recent version of the job input blob is used. All the earlier versions of the blob remain in the database until they are manually deleted. If you delete an input job blob, only the active version of the input job blob is deleted. The version which was prior to the deleted version becomes the new active version.

When you run a job, the CA Workload Automation AE agent downloads the active version of job's input blob from the database into a temporary file on the computer. This file is then passed into the standard input of the program that is executed by the job. When the job completes, the temporary file containing the input blob data is deleted. The blob in the database, however, is not deleted and remains as the active version for subsequent job runs.

Output and Error Job Blobs

Output and error job blobs store the program output and error messages of a running job. When you run a job, the CA Workload Automation AE agent creates temporary files on the computer that are used to capture the standard output and standard error messages from the program that was executed by the job. After the job has completed its run, the agent uploads the files containing the output data as blobs into the database, overwriting the existing files, and deletes the temporary files. An output job blob can be used as input by another job. An error job blob, on the other hand, cannot be used as input by another job.

Global Blobs

Global blobs are general purpose blobs in textual or binary data format. Like the CA Workload Automation AE global variables, they are referenced by a unique name. You can either upload the global blobs to the database using JIL or they can be uploaded by the CA Workload Automation AE agent, after a job has completed its run. After a global blob is created, it is available to any job as input. Global blobs remain in the database until they are deleted using JIL.

Manage Blobs Using JIL

The following section describes how to use JIL to do the following:

- Upload blobs to the database
- Delete blobs from the database

Note: For more information, see the *Reference Guide.*

Blob Attributes

The following table lists the subcommands and attributes associated with the definition or destruction of a blob:

Task	Subcommands	Attributes
Create input job blob	insert_job, update_job	blob_input or blob_file
Create input job blob	insert_blob	blob_input or blob_file
Delete job blob	delete_blob	blob_type

Task	Subcommands	Attributes
Create global blob	insert_glob	blob_mode, blob_input, or blob_file
Delete global blob	delete_glob	

The blob_input attribute lets you manually input the contents of a blob containing textual data. The blob_input attribute has the following format:

```
blob_input: <auto_blobt>textual data</auto_blobt>
```

Note: The textual data begins immediately after the auto_blobt XML-style open tag and may span multiple lines. JIL recognizes the end of the textual data when it reads the auto_blobt XML-style end tag. This implies that the literal character string *</auto_blobt>* cannot form part of the blob_input value. If you want to include this character string as part of the textual blob data, use the blob_file attribute.

The blob_file attribute allows the user to specify the location and name of a file on the computer that serves as the input job blob or global blob file. The blob_file attribute has the following format:

```
blob_file: filename
```

Note: If the blob_file attribute is used to specify an input job blob through the insert_job or insert_blob subcommand, the file is interpreted as a text-based file.

Create Input Job Blobs

To create an input job blob in the database using JIL, do the following:

- Upload an input job blob at the time of the definition of the associated job.
- Upload an input job blob after you have defined the job.

Note: Input job blobs are referenced by the name of the job.

To create an input job blob at the time of the definition of the associated job, use the insert_job JIL subcommand and specify either the blob_input or blob_file attributes, as follows:

```
insert_job: test_job_with_blob
job_type: cmd
command: sleep 60
machine: juno
owner: jerry@juno
std_in_file: $$blobt
blob_input: <auto_blobt>multi-lined text data for job blob
</auto_blobt>
```

or

```
blob_file: /test_job_with_blob_file.txt
```

To create an input job blob after you have defined the job, use the insert_blob JIL subcommand and specify either the blob_input or blob_file attributes, as follows:

```
insert_blob: test_job_with_blob
blob_input: <auto_blobt>multi-lined text data for job blob
</auto_blobt>
```

or

```
blob_file: /test_job_with_blob_file.txt
```

JIL interprets the file name that is specified in the blob_file attribute as a file that contains the textual data and performs a conversion of the new line character. JIL also displays the version number of the most recent input job blob.

Delete Job Blobs

You can use the JIL delete_blob subcommand to delete the following:

- Active version of the input job blob
- Output and error job blobs

You must specify whether to delete the job input or output blob data using the blob_type attribute.

Note: Job blobs are referenced by the name of the job. JIL displays the version number of the most recent job input blob.

To delete the most recent version of the input job blob, use the delete_blob JIL subcommand and specify the blob_type attribute with the value of *input,* as follows:

```
delete_blob: test_job_with_blob
blob_type: input
```

To delete the output and error job blobs, use the delete_blob JIL subcommand and specify the blob_type attribute with the value of *output*, as follows:

```
delete_blob: test_job_with_blob
blob_type: output
```

Create Global Blobs

You can use the JIL insert_glob subcommand to upload blobs containing textual or binary data.

As the global blobs are not associated with a job, you must do the following:

- Provide a unique identifier.
- Specify the mode of the blob data that is being used in the blob_mode attribute.

Note: If you use the insert_glob JIL subcommand using the same name as an existing global blob, the blob data is reinserted into the database. In this case, the original blob data is deleted and the new blob data takes its place.

To create a global blob containing textual data, use the insert_glob JIL subcommand and specify the blob_mode attribute with a value of *text* and either the blob_input or blob_file attributes, as follows:

```
insert_glob: my_text_global_blob
blob_mode: text
blob_input: <auto_blobt>multi-lined text data for job blob
</auto_blobt>
```

or

```
blob_file: /my_text_global_blob_file.txt
```

Note: JIL interprets the file name that is specified in the blob_file attribute as a file that contains textual data and performs a conversion of the new line character.

To create a global blob containing binary data, use the insert_glob subcommand and specify the blob_mode attribute with a value of *binary* and the blob_file attribute, as follows:

```
insert_glob: my_binary_global_blob
blob_mode: binary
blob_file: /my_binary_global_blob_file
```

Note: You cannot use the blob_input attribute to create a global blob that contains the binary data.

Delete Global Blobs

You can use the JIL delete_glob subcommand to delete the existing global blobs.

Note: You must provide a unique identifier because global blobs are not associated with a job.

To delete a global blob, use the delete_glob JIL subcommand and provide the name of an existing global blob, as follows:

```
delete_glob: my_global_blob
```

Use Blobs in Job Definitions

You can use the std_in_file, std_out_file, and std_err_file attributes of the JIL insert_job, update_job, or override_job subcommands to reference blobs in addition to files. Based on the keyword values you specify for these attributes, CA Workload Automation AE downloads a blob for input or uploads a job's output as blob to meet the job's needs.

The keywords are explained in the subsequent sections.

std_in_file Attribute

The keywords that are supported by the std_in_file attribute include the following:

$$blobt

Uses the input job blob of the current job as input and treats the blob data as textual data.

$$blob.<*job name*>

Uses the output job blob of the specified job as input and treats the blob data as binary data.

$$blobt.<*job name*>

Uses the output job blob of the specified job as input and treats the blob data as textual data.

$$glob.<*global blob name*>

Uses the specified global blob as input and treats the blob data as binary data.

$$globt.<*global blob name*>

Uses the specified global blob as input and treats the blob data as textual data.

Note: You cannot use the keyword *$$blob* to specify the use of the current job's input blob.

To define a job that uses the output blob of its previous run as input

1. Define the job so that the job's name is in the std_in_file attribute using either the $$blob.<*job name*> or $$blobt.<*job name*> keyword.
2. Apply a one-time override of the std_in_file attribute, so that the job reads from a local file on the computer on its first run.

std_out_file and std_err_file Attributes

The keywords that are supported by the std_out_file and std_err_file attributes include the following:

$$blob

Uploads the output or error of the current job as a job blob and treats the data as binary data.

$$blobt

Uploads the output or error of the current job as a job blob and treats the data as textual data.

$$glob.<*global blob name*>

Uploads the output or error of the current job as a global blob with the specified name and treats the data as binary data.

$$globt.<*global blob name*>

Uploads the output or error of the current job as a global blob with the specified name and treats the data as textual data.

Note:

- You cannot append data to an existing job or global blob.
- CA Workload Automation AE does not support the use of > or >> character strings in the std_out_file or std_err_file attributes.
- Existing blob data is overwritten with the new data after the job run is completed.

Generate Blob Reports Using Autorep

You can use the autorep utility to report on and download the input job blobs and global blobs. To export the job definition using the autorep –J <*jobname*> -q option includes exporting all versions of that job's input blob. If a download path is not specified, the contents of all input job blobs are displayed along with the job definition. Otherwise, autorep downloads the input blob to the specified directory and displays the input blob file names numbered by version along with the job definition. Reports generated against one or more global blobs are extracted in binary format unless otherwise specified using the –a command line parameter. If a download path is not specified, autorep downloads the global blob into a temporary directory.

Options specific to blob and glob data include the following:

-z *globname*

Specifies a glob name or mask whose contents are to be extracted. ALL may be specified to extract all globs. Wildcard characters % and _ are also supported.

-a

Specifies that the global blob can be downloaded as textual data.

-f *outdir*

Specifies the directory name where input job blobs or global blobs are extracted to. The default value is as follows:

- UNIX—The /tmp directory.
- Windows—The directory represented by the environment variable %TEMP%.

Note: For more information about autorep reports, job input, and global blobs, see the *Reference Guide.*

Example: Export Job Definition with Input Blobs

This example uses the autorep command to export a job definition:

```
autorep -J ALL -q
```

The output might resemble the following:

```
insert_job: test_job
job_type: cmd
command: cat
machine: juno
owner: jerry@ca
permission: gx,ge,wx
alarm_if_fail: 1
```

If the job has one or more input blobs tied to it, in addition to the job definition, the autorep command extracts each of the job blob definitions, and the output might resemble the following:

```
insert_job: test_job_with_blob   job_type: cmd
command: cat
machine: juno
owner: jerry@juno
permission:
std_in_file: $$blobt
alarm_if_fail: 1
/* -- test_job_with_blob:insert_blob #1 -- */
insert_blob: test_job_with_blob
blob_input: <auto_blobt>multi-lined text data for job blob 1
</auto_blobt>
/* -- test_job_with_blob:insert_blob #2 -- */
insert_blob: test_job_with_blob
blob_input: <auto_blobt> multi-lined text data for job blob 2
</auto_blobt>
/* -- test_job_with_blob:insert_blob #3 -- */
insert_blob: test_job_with_blob
blob_input: <auto_blobt> multi-lined text data for job blob 3
</auto_blobt>
```

You can also specify a location to download the blobs using the -f parameter as follows:

```
autorep -J ALL -q -f /myblobsdir
```

The output might resemble the following:

```
insert_job: test_job_with_blob   job_type: cmd
command: cat
machine: juno
owner: jerry@juno
permission:
```

```
std_in_file: $$blobt
alarm_if_fail: 1
/* -- test_job_with_blob:insert_blob #1 -- */
insert_blob: test_job_with_blob
blob_file: /myblobsdir/test_job_with_blob_1.txt
/* -- test_job_with_blob:insert_blob #2 -- */
insert_blob: test_job_with_blob
blob_file: /myblobsdir/test_job_with_blob_2.txt
/* -- test_job_with_blob:insert_blob #3 -- */
insert_blob: test_job_with_blob
blob_file: /myblobsdir/test_job_with_blob_3.txt
```

Example: Generate a Report for All Global Blobs

This example generates a report that downloads the contents of all global blobs to the location */myblobsdir* as binary data:

```
autorep -z ALL -f /myblobsdir
```

The report might resemble the following:

```
Glob Name        File Name

____________  _____________________________
MYGLOB           /myblobsdir/MYGLOB
REPORT_CHART     /myblobsdir/REPORT_CHART
ARCHIVED_DATA    /myblobsdir/ARCHIVED_DATA
JOB_SNAPSHOT     /myblobsdir/JOB_SNAPSHOT
```

This example generates a report that downloads the contents of all global blobs to the location */myblobsdir* as text data:

```
autorep -z ALL -f /myblobsdir -a
```

The report might resemble the following:

```
Glob Name        File Name

___________  ____________________________________
MYGLOB           /myblobsdir/MYGLOB.txt
REPORT_CHART     /myblobsdir/REPORT_CHART.txt
ARCHIVED_DATA    /myblobsdir/ARCHIVED_DATA.txt
JOB_SNAPSHOT     /myblobsdir/JOB_SNAPSHOT.txt
```

Chapter 23: Cross-Instance Scheduling

This section contains the following topics:

Bi-Directional Scheduling

CA Workload Automation AE supports *bi-directional scheduling*, which lets you start jobs from remote machines (inbound) or submit jobs on remote machines (outbound).

With *inbound job scheduling*, CA Workload Automation AE acts as an agent and accepts job submissions from remote machines or other scheduling managers (such as CA Jobtrac Job Management and CA Workload Automation SE). The jobs are defined and run on the CA Workload Automation AE instance that is acting as an agent.

With *outbound job scheduling*, CA Workload Automation AE acts as a scheduling manager and sends job submissions to remote machines. The jobs are defined on the CA Workload Automation AE instance that is acting as a scheduling manager. The jobs run on the remote machine or other scheduling manager.

For example, a Linux Oracle instance can initiate jobs in a Windows Microsoft SQL Server instance, or a Windows Microsoft SQL Server instance can initiate jobs in a Solaris Oracle instance. You can add additional instances, such as Solaris Sybase, AIX Oracle, or HP Oracle instance, to the environment.

The CA Workload Automation AE cross-platform interface controls the bi-directional scheduling mode. You can configure the cross-platform interface to enable the following modes:

- Outbound job scheduling
- Inbound and outbound job scheduling (bi-directional scheduling)
- No cross-platform scheduling (the default)

Note: There are no restrictions on platforms, event servers, or number of instances when running in bi-directional scheduling mode.

CA Workload Automation AE Cross-Instance Job Dependencies

A CA Workload Automation AE *instance* is one licensed version of CA Workload Automation AE software running as a server and as one or more clients, on one or more computers. An instance uses its own scheduler, one or more application servers, and event server, and operates independently of other instances.

Different instances can run from the same executables and can have the same value for $AUTOSYS. However, each instance must have different values for $AUTOUSER and $AUTOSERV. Different instances can also be run on the same computer.

Multiple CA Workload Automation AE instances are not connected, but they can communicate with one another. This communication lets you schedule workload across instances in your enterprise. You can define jobs that have dependencies on jobs running on other instances (*cross-instance job dependencies*). A CA Workload Automation AE job with these dependencies conditionally starts based on the status of the job on the other instance. In this situation, your instance's scheduler acts as a client and issues sendevent commands to the external instance. The other instance's application server processes the sendevent request and stores the dependency request or status update in its database.

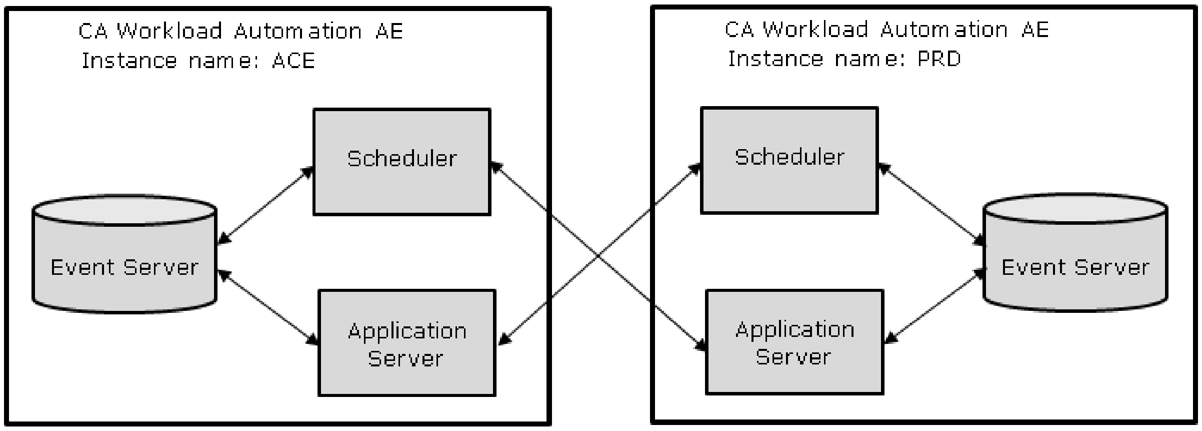
You can also manually send events from one instance to another.

Note: Before you can submit jobs on other CA Workload Automation AE instances, you must define the instances to each other. For more information about configuring CA Workload Automation AE to support cross-instance scheduling, see the *UNIX Implementation Guide* or *Windows Implementation Guide*.

How Cross-Instance Job Dependencies are Processed

You can associate jobs with more than one CA Workload Automation AE instance. For example, you can define a job to conditionally start based on the status of a job on another instance.

The following illustration shows two instances exchanging cross-instance job dependencies:



Note: If instance ACE’s application server runs on port 9001 and instance PRD’s application server runs on port 9002, you must verify that both ports are configured using SSA on both machines. Otherwise, communication with the remote application server will fail.

The CA Workload Automation AE event server tracks cross-instance job dependencies as follows:

Each time a job definition with a cross-instance job dependency is submitted to the database, the event server does the following:

- Makes an entry in the ujo_ext_job table of the issuing instance. The entries in this table specify the status of jobs in other instances that the issuing instance has an interest in.
- Makes an entry in the ujo_req_job table of the receiving instance. The entries in this table specify the jobs defined as job dependencies in a job definition on the issuing instance.

The jobs are entered in the ujo_ext_job and ujo_req_job tables using the following syntax:

job_name^INSTANCE_NAME

For example, jobB^PRD indicates a job named jobB on the PRD instance.

The use of multiple databases is independent of instances using cross-instance dependencies. You can have multiple instances that each use dual event servers.

When CA Workload Automation AE encounters a cross-instance dependency, it sends an EXTERNAL_DEPENDENCY event from the requesting instance. The following process occurs when one instance cannot send status updates (events) to the other instance:

- An INSTANCE_UNAVAILABLE alarm is issued.
- The ujo_asext_inst table is updated to indicate that the external instance is offline.
- While the job continues to run, all external events to be sent to the external instance are stored in the ujo_ext_event table.
- The local instance periodically tries to connect to the external instance.
- When local instance reconnects successfully to the external instance, all the events are sent to the external instance, and the external events are deleted from the ujo_ext_event table.

Types of External Instances

To use external job dependencies, the scheduling manager or remote machine must be defined as an *external instance* in the CA Workload Automation AE database.

When you define the external instance, you must identify the type using the xtype JIL attribute. Options are the following:

xtype: a

Indicates that the external scheduling manager is a CA Workload Automation AE application server instance.

xtype: c

Indicates that CA AutoSys WA Connect Option is installed with the external scheduling manager. CA AutoSys WA Connect Option can be installed on the mainframe and supports cross-instance jobs and job dependencies. It lets you submit job requests to and receive job submissions from the following mainframe scheduling managers:

- CA Jobtrac Job Management
- CA Scheduler Job Management
- CA Workload Automation SE

The CA Workload Automation AE scheduler uses CAICCI to communicate with CA AutoSys WA Connect Option.

xtype: u

Indicates that CA UJMA is installed with the external scheduling manager or on the remote machine. CA UJMA can be installed on the mainframe, UNIX, and Windows. It lets you submit job requests to the remote machine where CA UJMA is installed. It lets you submit job requests to and receive job submissions from the following scheduling managers:

- CA Job Management Option
- CA Jobtrac Job Management
- CA Scheduler Job Management
- CA Workload Automation SE

The CA Workload Automation AE scheduler uses CAICCI to communicate with CA UJMA.

Note: Unlike CA AutoSys WA Connect Option, CA UJMA does not let you define cross-instance job dependencies on the mainframe. To define cross-instance job dependencies on the mainframe, you must install CA AutoSys WA Connect Option on the same computer as the mainframe scheduling manager.

xtype: e

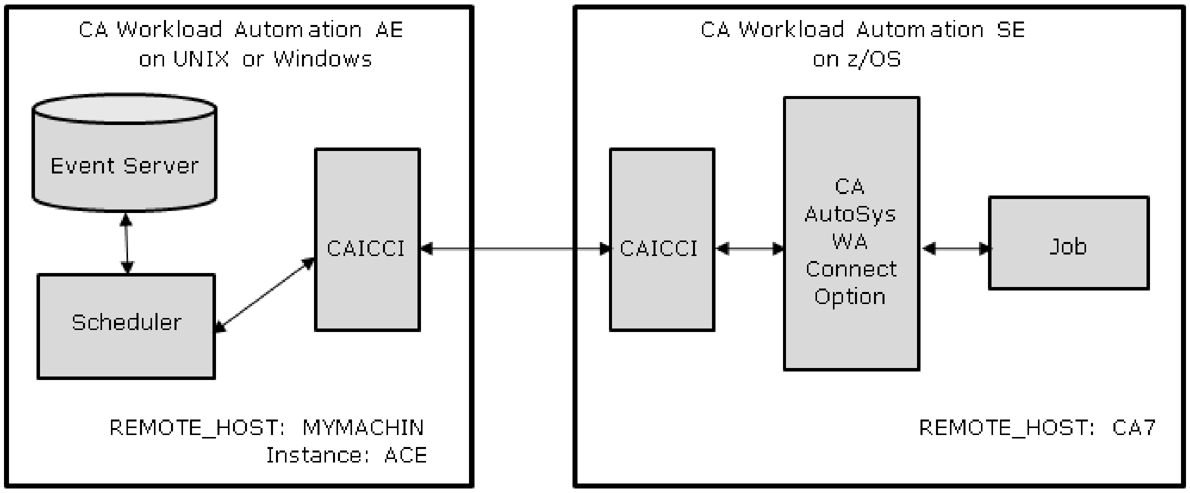
Indicates that the external scheduling manager is CA Workload Automation EE. You can define cross-instance job dependencies.

Note: Bi-directional scheduling is currently not supported between CA Workload Automation AE and CA Workload Automation EE.

Creating Cross-Instance Job Dependencies Using CA AutoSys WA Connect Option

CA Workload Automation AE jobs can have dependencies on jobs managed by an CA AutoSys WA Connect Option and a CA scheduling manager running on the mainframe. The mainframe scheduling manager uses CA AutoSys WA Connect Option and CAICCI to communicate with CA Workload Automation AE. The CA Workload Automation AE scheduler also uses its cross-platform scheduling interface for communication.

For example, the following illustration shows a CA Workload Automation AE job defined on a UNIX or Windows computer. The job's starting condition is the successful completion of a job running on the mainframe.



CA Workload Automation AE jobs can be dependent on the status of external jobs managed by CA AutoSys WA Connect Option, and external jobs can be dependent on the status of CA Workload Automation AE jobs. CA Workload Automation AE uses the following process to create cross-instance dependencies:

1. The CA Workload Automation AE scheduler sends a request for the status of a CA AutoSys WA Connect Option job.
2. CA AutoSys WA Connect Option registers the request.
3. The CA AutoSys WA Connect Option job runs on the mainframe.
4. CA AutoSys WA Connect Option sends the job status to the CA Workload Automation AE scheduler.
5. The CA Workload Automation AE scheduler communicates the status to the event server.
6. The CA Workload Automation AE scheduler processes the status and starts the job that is dependent on the completion of the CA AutoSys WA Connect Option job, if appropriate.

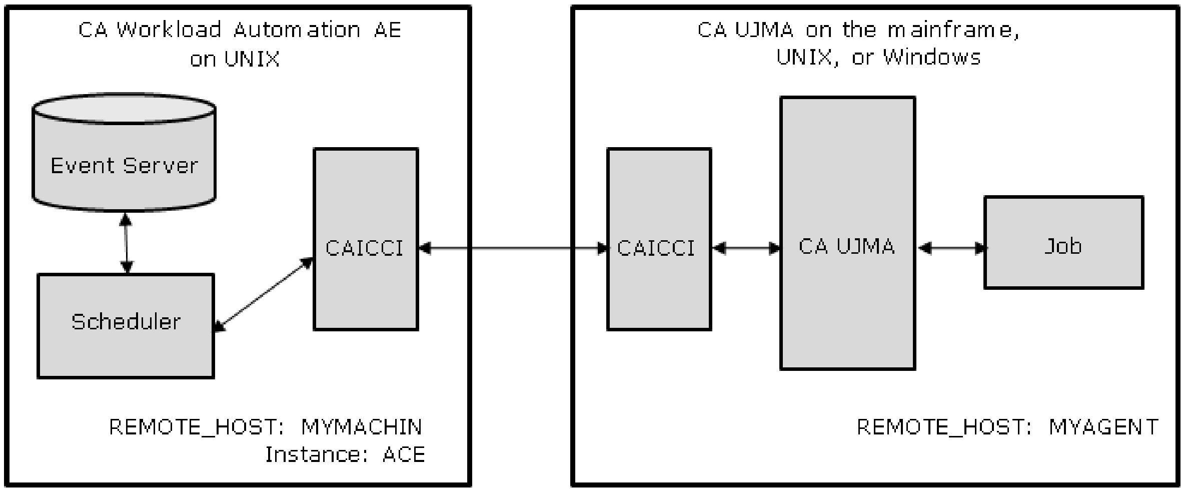
Note: For more information about the CAICCI components, see the *UNIX Implementation Guide* or *Windows Implementation Guide*.

Submitting a Job To and From Another Computer Using CA UJMA

CA Workload Automation AE can schedule jobs on a computer that has CA UJMA installed on it. As job submission requests are processed, the scheduler log file records events sent between CA Workload Automation AE and CA UJMA. The following descriptions help you understand the events recorded in the scheduler log file.

Note: CA Workload Automation AE can directly schedule jobs on a computer that is running a supported workload automation agent. This topic only discusses the communication with CA UJMA.

The following diagram shows the components involved in the communication:



The CA Workload Automation AE scheduler communicates directly with CA UJMA using CAICCI and the CA Workload Automation AE Cross-Platform Interface. The communication components running on the CA UJMA computer receive information from the CA Workload Automation AE scheduler and pass it to CA UJMA. Similarly, CA UJMA passes information through the communication components to the CA Workload Automation AE scheduler.

CA UJMA does the following:

- Receives job requests from one or more CA scheduling managers (such as CA Job Management Option (JMO), CA Workload Automation AE, and CA Workload Automation SE). CA UJMA initiates the requested program, script, JCL, or other unit of work. If you are scheduling to the mainframe, the command or program to initiate is the job name of the job as defined in the mainframe scheduling system.
- Collects status information about job runs.
- Sends status information to the requesting scheduling manager.

Note: Unlike CA AutoSys WA Connect Option, CA UJMA does not support cross-instance job dependencies.

The following process is used to start a job on a CA UJMA computer:

- You define a job on CA Workload Automation AE that specifies the job to start on the CA UJMA computer. The mainframe job to start (specified in the command attribute of the CA Workload Automation AE job definition) must be a named job known to the CA Workload Automation AE scheduler. The mainframe job to start cannot be a command or script.
- The CA Workload Automation AE scheduler sends the job information to CA UJMA.
- The job changes to STARTING status.
- CA UJMA starts the job and sends an event to the scheduler. The event sent is JOBINITU if the job started or JOBFAILU if the job could not start.
- The scheduler converts the JOBINITU event to RUNNING, puts it in the database, and updates the job's status to RUNNING. If CA UJMA sent a JOBFAILU event, the scheduler converts the event to FAILURE and processes it accordingly.
- If the job completes successfully, CA UJMA sends a JOBTERMU event to the scheduler.
- The scheduler converts the JOBTERMU event to SUCCESS, FAILURE, or TERMINATED based on the exit code of the job. If the job exited with a normal end of job code (EOJ) the scheduler converts JOBTERMU to SUCCESS or FAILURE. If the job exited with an abnormal end of job code (AEOJ), the scheduler converts JOBTERMU to TERMINATED.

CA Workload Automation AE can also receive job submission requests from a CA UJMA computer. The job to start must be a job defined on CA Workload Automation AE and cannot be a command or script. If CA Workload Automation AE receives a job submission request from the mainframe, the job to run (specified by the SUBFILE parameter of the mainframe job) must be defined as a valid job on the CA Workload Automation AE computer.

For the CA Workload Automation AE scheduler to communicate with CA UJMA computers, the scheduler must convert CA Workload Automation AE-based events to events that CA UJMA can interpret. Similarly, the CA Workload Automation AE scheduler must convert events returned from CA UJMA back to events the scheduler can interpret.

The following table lists the CA Workload Automation AE and CA UJMA events:

Operation	**CA Workload Automation AE**	**CA UJMA**
Starting a job	STARTJOB	SUBMITU
Job has started and is running	RUNNING	JOBINITU

Operation	CA Workload Automation AE	CA UJMA
Job has terminated successfully with an exit code	SUCCESS or FAILURE	JOBTERMU
Job has failed to start	FAILURE	JOBFAILU

Unsupported Attributes for CA AutoSys WA Connect Option or CA UJMA Jobs

The following table lists attributes that are not supported for CA AutoSys WA Connect Option and CA UJMA jobs. If you specify these attributes, they are ignored.

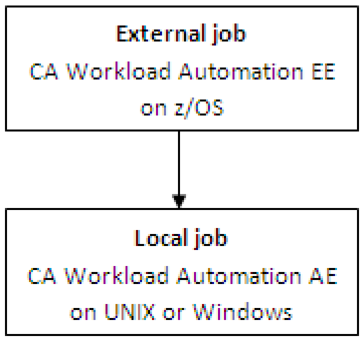
JIL Attribute	CA WCC Field
chk_files	File system check
heartbeat_interval	Heartbeat interval
job_load	Job load
job_terminator	Terminate on failure of containing box
job_type:f	File Watcher (FW) job in either Quick Edit or Application Editor
n_retrys	Times to restart job after failure
priority	Priority
profile	Profile
std_err_file	Standard error file
std_in_file	Standard input file
std_out_file	Standard output file
term_run_time	Minutes to wait before terminating
watch_file	File(s) to watch
watch_file_min_size	Minimum file size
watch_interval	Watch interval

Note: Computers managed by CA UJMA computers cannot be part of a virtual machine. The max_load and factor attributes are also not supported when defining an external CA UJMA instance.

How Job Dependencies are Processed Using CA Workload Automation EE

You can define job dependencies between CA Workload Automation AE and an external CA Workload Automation EE instance running on the mainframe. As job dependencies are processed, the scheduler log file records events communicated between CA Workload Automation AE and CA Workload Automation EE. The following descriptions help you understand the events recorded in the scheduler log file.

A CA Workload Automation AE job can depend on another job on CA Workload Automation EE as shown in the following diagram:



The job dependency is processed as follows:

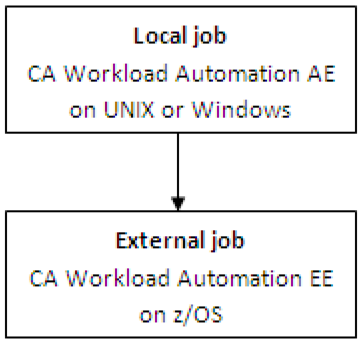
- The job dependency is specified in the condition attribute of the local CA Workload Automation AE job definition. For example, the following job depends on the success of the ZOS01 job defined on the CA Workload Automation EE instance named MSR:

  ```
  insert_job: MSR1
  job_type: CMD
  command: sleep 10
  machine: localhost
  owner: root@hostname.ca.com
  condition: s(ZOS01^MSR)
  ```

- When the CA Workload Automation AE job is created or updated, an EXTERNAL_DEPENDENCY event is created in the CA Workload Automation AE database. The event contains the information in the condition attribute.
- The CA Workload Automation AE scheduler sends a job dependency request to CA Workload Automation EE.
- As the job runs on the external instance, CA Workload Automation EE sends status updates to CA Workload Automation AE.

- The CA Workload Automation AE scheduler updates the ujo_ext_job table with the status updates from CA Workload Automation EE.
- When the CA Workload Automation AE job is deleted or its condition attribute is changed, an EXTERNAL_DEPENDENCY event is created in the CA Workload Automation AE database. The event contains the information required to remove the external job dependency from the ujo_ext_job table.
- The CA Workload Automation AE scheduler sends a request to CA Workload Automation EE to deregister the job dependency on the external instance.

A job on CA Workload Automation EE can depend on a CA Workload Automation AE job as shown in the following diagram:


Local job
CA Workload Automation AE on UNIX or Windows
External job
CA Workload Automation EE on z/OS

The job dependency is processed as follows:

- When a new generation of an application is started in the external instance, CA Workload Automation EE sends a job dependency request to CA Workload Automation AE for a single run of a job.
- The CA Workload Automation AE scheduler creates an EXTERNAL_DEPENDENCY event on its behalf.
- The CA Workload Automation AE scheduler updates the ujo_req_job database table to record that a CA Workload Automation EE job is dependent on the local job.
- As the job runs on the local CA Workload Automation AE instance, the CA Workload Automation AE scheduler sends status updates to CA Workload Automation EE.
- When the job has its run, the CA Workload Automation AE scheduler deletes the dependent job entry from the ujo_req_job table.

The following process occurs when CA Workload Automation AE cannot send status updates (events) to CA Workload Automation EE:

- An INSTANCE_UNAVAILABLE alarm is issued.
- The ujo_asext_inst table is updated to indicate that the external instance is offline.
- While the job continues to run, all events to be sent to the external instance are stored in the ujo_ext_event table.
- The CA Workload Automation AE scheduler periodically tries to connect to CA Workload Automation EE.
- When the CA Workload Automation AE scheduler successfully re-connects to CA Workload Automation EE, all the events are sent to CA Workload Automation EE, and the events are deleted from the ujo_ext_event table.

To communicate with CA Workload Automation EE, the CA Workload Automation AE scheduler converts CA Workload Automation AE-based events to events that CA Workload Automation EE can interpret. Similarly, the scheduler must convert events returned from CA Workload Automation EE back to events that the scheduler can interpret. All converted events are recorded in the scheduler log file to indicate that the events are from an external instance.

The following table lists the CA Workload Automation AE and CA Workload Automation EE events:

CA Workload Automation AE Event	CA Workload Automation EE Event
STARTING	READY
RUNNING	EXEC
SUCCESS	COMPLETED
FAILURE	FAILED
TERMINATED	FAILED with a Job Terminated status
ON_ICE	COMPLETED with a BYPASSED status
INACTIVE	HELD

Notes:

- CA Workload Automation AE communicates directly with CA Workload Automation EE. You do not have to install additional products to integrate the scheduling managers.
- Bi-directional scheduling is currently not supported between CA Workload Automation AE and CA Workload Automation EE. You cannot submit jobs to or receive job requests from CA Workload Automation EE at this time.

Cross-Platform Scheduling

Cross-platform scheduling lets you schedule and reroute jobs between CA Workload Automation AE and other machines running on different platforms, including mainframe.

To use cross-platform scheduling, required components must be installed on the CA Workload Automation AE computer and on the external machine that CA Workload Automation AE works with. The scheduling manager or remote machine must also be defined as an *external instance* in the CA Workload Automation AE database.

Note: Before you can submit jobs on other scheduling managers, you must activate the CA Workload Automation AE cross-platform interface and define the scheduling manager as an external instance on CA Workload Automation AE. For more information about configuring CA Workload Automation AE to support cross-platform scheduling, see the *UNIX Implementation Guide* or *Windows Implementation Guide*.

Submitting a Job To and From the Mainframe Using CA AutoSys WA Connect Option

CA Workload Automation AE lets you schedule workload across distributed and mainframe platforms. You can submit jobs on the mainframe and receive job submission requests from the mainframe. Depending on the mainframe scheduling manager, the mainframe must have the following software installed on it:

- CA Jobtrac Job Management and CA AutoSys WA Connect Option
- CA Scheduler Job Management and CA AutoSys WA Connect Option
- CA Workload Automation EE
- CA Workload Automation SE and CA AutoSys WA Connect Option

CA Workload Automation AE and the mainframe scheduling manager use CAICCI to communicate.

When submitting a job to the mainframe, the following process occurs:

- The CA Workload Automation AE scheduler on the distributed platform interrogates the start job request, processes it, and transmits it to the scheduling manager installed on the mainframe.
- After the scheduling manager receives the request on the mainframe, it submits and tracks the mainframe job. The job submission and tracking is completed in one of the following ways:
  - Directly by the XPS-enabled mainframe scheduling product
  - Through CA AutoSys WA Connect Option

When CA Workload Automation AE accepts a job submitted from a mainframe system, CA Workload Automation AE reports its status back to the scheduling manager on the mainframe.

Note: For more information about submitting the job from an external mainframe scheduling manager, see the appropriate product documentation.

Cross-Platform Interface Messages Logged for CA UJMA

On UNIX, all messages produced by the cross-platform interface are written to the CA Workload Automation AE scheduler log, which is located in the $AUTOUSER/out directory.

On Windows, all messages produced by the cross-platform interface are written to the CA Workload Automation AE scheduler log, which is located in the %AUTOUSER%\out directory.

The following message indicates that the scheduler has transferred a job to the cross-platform interface for submission to a CA UJMA computer:

```
CAUAJM_I_10073 AutoSys --> Cross Platform Interface:
machine=machine_name     job_name=job_name
```

machine_name

Identifies the CA UJMA computer to which the job is being submitted.

job_name

Identifies the CA Workload Automation AE job name as defined to the event server.

The following message indicates that an event status has been received from CA UJMA. The event status is converted to the appropriate CA Workload Automation AE event status and inserted in the event server.

```
CAUAJM_I_40263 EVENTU: event_name
EXITCODE: exitcode/dbcode JOB: job_name
```

event_name

Identifies one of the following events:

JOBINITU

Indicates that a job has started on a CA UJMA computer.

JOBTERMU

Indicates that a job has completed on a CA UJMA computer.

JOBFAILU

Indicates that a job has failed to start on a CA UJMA computer.

SUBMITU

Indicates that a job has been submitted to CA Workload Automation AE.

exitcode/dbcode

Identifies the actual job exit code returned by CA UJMA.

job_name

Identifies the CA Workload Automation AE job name as defined to the event server.

Define an External Instance

To use cross-instance scheduling, you must define the external instance to CA Workload Automation AE. The external instance can be another CA Workload Automation AE instance or a scheduling manager running on another platform.

To define an external instance

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.
2. Specify the following definition:

   ```
   insert_xinst: instance_name
   xtype: a | c | u | e
   xmachine: host_name
   ```

   xtype: a | c | u | e

   Specifies the external instance type. Options include the following:
   - a—Identifies a remote CA Workload Automation AE application server instance.
   - c—Identifies a CA AutoSys Workload Automation Connect Option instance.
   - u—Identifies a CA Universal Job Management Agent or CA NSM instance.
   - e—Identifies a CA Workload Automation EE instance.
3. (CA Workload Automation AE instances only) Specify the following additional attributes:

   ```
   xcrypt_type: NONE | DEFAULT | AES
   xkey_to_manager: encryption_key
   xport: port_number
   ```
4. (CA Workload Automation EE instances only) Specify the following additional attributes:

   ```
   xmanager: manager_name
   xport: port_number
   xcrypt_type: NONE | AES
   xkey_to_manager: encryption_key
   ```
5. Do *one* of the following:
   - Enter **exit** if you are using interactive mode.
   - Redirect the script to the jil command if you are using a script.

   The insert_xinst subcommand is issued and the specified external instance is defined.

Notes:

- For CA Workload Automation EE, the xcrypt_type value must match the encryption type specified in the AGENTDEF data set.
- For more information about the syntax for the insert_xinst subcommand and related attributes, see the *Reference Guide*.

Example: Define a CA Workload Automation EE Instance

This example defines the CA Workload Automation EE instance named CYB to CA Workload Automation AE. CYB runs on the CYBHOST computer at port 7550. The CM manager name is unique to this CA Workload Automation EE instance. The CA Workload Automation AE scheduler communication alias is defined in the AGENTDEF data set on CA Workload Automation EE. CA Workload Automation EE is also configured to point to the CA Workload Automation AE computer on the auxiliary listening port.

```
insert_xinst: CYB
xtype: e
xmanager: CM
xmachine: CYBHOST
xcrypt_type: NONE
xport: 7550
```

Update an External Instance

You can update an external instance defined in CA Workload Automation AE. The external instance can be another CA Workload Automation AE instance or a scheduling manager running on another platform. You can update the instance definition when the connection information changes. The connection information must be accurate so that the scheduling managers can communicate with each other.

To update an external instance

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.
2. Specify the following JIL subcommand:

   ```
   update_xinst: instance_name
   ```

3. Specify the following attributes as appropriate:

   ```
   xmachine: host_name
   xcrypt_type: NONE | DEFAULT | AES
   xkey_to_manager: encryption_key
   xport: port_number
   xmanager: manager_name
   ```

4. Do *one* of the following:

   - Enter **exit** if you are using interactive mode.
   - Redirect the script to the jil command if you are using a script.

   The update_xinst subcommand is issued and the specified external instance is updated.

Note: For more information about the syntax for the update_xinst subcommand and related attributes, see the *Reference Guide*.

Example: Update a CA Workload Automation EE Instance

Suppose that an external instance is defined on CA Workload Automation AE for the CA Workload Automation EE instance named CYB. This example updates the external instance definition so that CYB points to the CYBHOST2 computer at port 7551.

```
update_xinst: CYB
xmachine: CYBHOST2
```

Delete an External Instance

You can delete an external instance that you no longer use from CA Workload Automation AE.

To delete an external instance

1. Do *one* of the following:
   - Issue JIL in interactive mode.
   - Open a JIL script in a text editor.
2. Specify the following JIL subcommand:

   ```
   delete_xinst: instance_name
   ```
3. Do *one* of the following:
   - Enter **exit** if you are using interactive mode.
   - Redirect the script to the jil command if you are using a script.

   The insert_xinst subcommand is issued and the specified external instance is deleted from CA Workload Automation AE.

Note: For more information about the syntax for the delete_xinst subcommand, see the *Reference Guide*.

Example: Delete a CA Workload Automation EE Instance

This example deletes the definition of the CA Workload Automation EE instance named CYB.

```
delete_xinst: CYB
```

Start a Job on an External CA Workload Automation AE Instance

CA Workload Automation AE instances are not connected, but they can communicate with one another. You can send a STARTJOB event directly from one instance to another. This helps you integrate your workload across CA Workload Automation AE instances in your enterprise.

To start a job on an external CA Workload Automation AE instance, enter the following command at the UNIX operating system prompt or the Windows instance command prompt:

```
sendevent -E STARTJOB -J job_name -S autoserv
```

job_name

Specifies a job defined on the *autoserv* instance.

autoserv

Specifies the instance's unique, capitalized three-character identifier.

Example: ACE

The sendevent command is issued and the job request is submitted on the external CA Workload Automation AE instance.

Notes:

- To use the sendevent -S option, the instance from where you issue the sendevent command must have a client installed for the external instance. For more information about configuring cross-instance communication, see the *UNIX Implementation Guide* or *Windows Implementation Guide*.
- For more information about the syntax for the sendevent command and related attributes, see the *Reference Guide*.

Define a Job to Run on an External Instance

From your local CA Workload Automation AE instance, you can submit a job to run on an external instance. The external instance is a different scheduling manager installed on another platform, including mainframe. Submitting external jobs helps you integrate workload across platforms in your enterprise.

Note: Before you can submit jobs on other scheduling managers, you must activate the CA Workload Automation AE cross-platform interface and define the scheduling manager as an external instance on CA Workload Automation AE. For more information about configuring CA Workload Automation AE to support cross-platform scheduling, see the *UNIX Implementation Guide* or *Windows Implementation Guide*.

To define a job to run on an external instance

1. Insert a job and specify the following attributes in the definition:

   job_type: CMD

   Specifies that the job type is Command.

   machine

   Specifies the name of the external instance.

   Note: The external instance must be defined on CA Workload Automation AE using the insert_xinst subcommand.

   owner

   Specifies the user on the external instance that the job runs under. The owner must be specified as *owner@machine.*

   CA UJMA Note: The specified owner must have an account on the external CA UJMA computer. The account must match the owner name exactly, and the owner (user ID and password) must be defined using the autosys_secure command.

command

Specifies the job to run on the external instance when all the starting conditions are met.

CA UJMA Limits (UNIX or Windows): Where the operating system permits, CA UJMA job names on distributed systems can contain up to 64 alphanumeric characters and can contain both uppercase and lowercase characters. You cannot use blank spaces and tab characters.

CA UJMA Limits (mainframe): CA UJMA job names on the mainframe must follow these guidelines:

- The first character of a job name must be an uppercase letter (A-Z), a pound sign (#), an at sign (@), or a dollar sign ($).
- The remaining characters in the job name can be any combination of uppercase letters (A-Z), numbers (0-9), pound signs (#), at signs (@), and dollar signs ($).
- All letters (A-Z) must be in uppercase.
- Job names can be up to eight characters in length.

2. (Optional) Specify common attributes that apply to all jobs.

   The external job is defined.

Note: For more information about the syntax for the JIL attributes, see the *Reference Guide*.

Example: Define a Job to Run on an AS/400 Computer

This example defines a command job to run on an AS/400 computer.

```
insert_job: as400_a1
job_type: CMD
command: DLYJOB DLY(15)
machine: usprncax
owner: user1@usprncax
permission: gx,wx
date_conditions: 1
days_of_week: all
start_mins: 30
```

Example: Define a Job to Run Through CA AutoSys WA Connect Option

This example defines a command job to run in CA Workload Automation SE through CA Workload Automation AE Connect.

```
insert_job: ca71
job_type: CMD
command: auto_cnct -a A87SOENF -j RYAKEJ01 -c RUN -p SCHEDULE=RYAKE01 -s CA7
machine: A87SOENF
owner: user1@A87SOENF
permission: gx,wx
date_conditions: 1
days_of_week: all
start_mins: 45
```

Example: Define a Job to Run Directly in CA Workload Automation SE

This example defines a command job to run directly in CA Workload Automation SE.

```
insert_job: ca72
job_type: CMD
command: RYAKEJ01
machine: A87SOENF
owner: user1@A87SOENF
permission: gx,wx
date_conditions: 1
days_of_week: all
start_mins: 45
```

Example: Define a Job to Run in Another CA Workload Automation AE Instance

This example defines a command job to run in another CA Workload Automation AE instance.

```
insert_job: ca72
job_type: cmd
command: job_in_other_instance
machine: othermachine
owner: user1@othermachine
permission: gx,wx
date_conditions: 1
days_of_week: all
start_mins: 45
```

This example uses the following configuration:

- In the machine definition for othermachine, *machine_type* was set to u, indicating a machine that runs CA UJMA.
- The cross-platform interface was activated on the server (where job ca72 is being submitted) by setting the CrossPlatformScheduling parameter to 1 (run jobs directly on a CA UJMA agent). If this instance will receive job submissions from any scheduling manager in the enterprise, set the CrossPlatformScheduling parameter to 2 (enable bi-directional scheduling support) instead.
- The CrossPlatformScheduling parameter for the machine othermachine was set to 2 (enable bi-directional scheduling support).

Example: Define a Job to Run on a UNIX Computer

This example defines a command job to run on a UNIX computer.

```
insert_job: soljob1
command: “@SYS$LOGIN:SCHEDULE_WAIT.COM “
machine: mysolaris
owner: user1@mysolaris
max_exit_success: 1
```

Note: A job that runs successfully on a UNIX computer returns an exit code of 1. By default, CA Workload Automation AE interprets an exit code of 1 as a failure unless the max_exit_success attribute is properly set in the job definition.

Define a Cross-Instance Job Dependency

You can define a job to be dependent on another job that runs on an external instance. The external instance can be another CA Workload Automation AE instance or a scheduling manager running on another platform.

To define a cross-instance job dependency, add the following attribute to your job definition:

```
condition: status(JOB_NAME^INS) [AND|OR status(JOB_NAME^INS)...]
```

status

Specifies the status that the external job must have before your job starts. Options are the following:

- done
- failure
- notrunning
- success
- terminated

JOB_NAME

Specifies the name of the external job that the local job depends on.

Limits: The names of jobs specified as job dependencies between CA Workload Automation AE and CA AutoSys WA Connect Option must follow these guidelines:

- The first character of a job name must be an uppercase letter (A-Z), a pound sign (#), an at sign (@), or a dollar sign ($).
- The remaining characters in the job name can be any combination of uppercase letters (A-Z), numbers (0-9), pound signs (#), at signs (@), and dollar signs ($).
- All letters (A-Z) must be in uppercase.
- Job names can be up to eight characters in length.

INS

Specifies the name of the external instance that *JOB_NAME* runs on. The name must be three uppercase alphanumeric characters. The first character must be a letter (A-Z).

Notes:

- You can specify multiple job dependencies in a definition.
- For detailed information about the syntax for the condition attribute, see the *Reference Guide*.

Example: Specify a Cross-Instance Job Dependency

This example defines a job that runs only when the following starting conditions are met:

- jobA on the same instance returns SUCCESS
- jobB on the CA Workload Automation AE instance PRD returns SUCCESS

```
insert_job: dep_job1
machine: localhost
job_type: CMD
command: sleep 100
condition: success(jobA) AND success(jobB^PRD)
```

Generate a Report on an External Instance

You can generate a report on an external instance to verify the instance configuration is correct. Generating a report is helpful when you want to check the status of an external instance during failover. The event details in a report can also help you determine whether the correct events are waiting for updates from the external instance.

To generate a report on an external instance, enter the following command at the UNIX operating system prompt or the Windows instance command prompt:

```
autorep –X external_instance [-q]
```

-X *external_instance*

Specifies the external instance you want to generate a report on.

-q

(Optional) Generates a query report, which contains the current job or machine definition.

The autorep command is issued and the report is generated and displayed.

Note: For more information about the autorep command, see the *Reference Guide*.

Example: Generate a Report on an External Instance

This example generates a report on the external machine named PRD.

```
autorep -X PRD

Name Type Server                        Port

____ ____ _____________________________ ____
PRD  a    nyc-wall-04                   9001
```

Example: Generate a Report on an External Instance

This example generates a query report on the external machine named PRD. The report displays the machine definition.

```
autorep -X PRD -q
/* ----------------- PRD ----------------- */
insert_xinst: PRD
xtype: a
xmachine: nyc-wall-04
xport: 9001
xcrypt_type: DEFAULT
```

Example: Export External Instance Definitions

This example uses the autorep command to export all external instance definitions.

```
autorep -X ALL -q
```

The output might resemble the following:

```
/* ----------------- CCT ----------------- */

insert_xinst: CCT
xtype: c
xmachine: WACNCTHOST


/* ----------------- NSM ----------------- */

insert_xinst: NSM
xtype: u
xmachine: NSMHOST


/* ----------------- WAE ----------------- */

insert_xinst: WAE
xtype: a
xmachine: WAAEHOST
```

```
xport: 9001
xcrypt_type: DEFAULT


/* ----------------- WEE ----------------- */

insert_xinst: WEE
xtype: e
xmachine: WAEEHOST
xport: 7550
xcrypt_type: NONE
xmanager: WAAEMGR
```

Example: Generate a Report for All External Instances

This example uses the autorep command to generate a report of all external instances:

```
autorep -X ALL
```

The report might resemble the following:

```
Name Type Server                        Port
____ ____ _____________________________ ____
CCT  c    WACNCTHOST                    0
NSM  u    NSMHOST                       0
WAE  a    WAAEHOST                      9001
WEE  e    WAEEHOST                      7550
```

Chapter 24: Monitoring and Reporting Jobs

This section contains the following topics:

Monitors and Reports

Monitors provide a real-time view of the system. *Reports* (sometimes called browsers) let you use a variety of reporting views to examine historical information about job executions.

Monitors and reports are simply applications that run against the database. Because all information is in the database, monitors and reports that retrieve information from the database provide a complete picture of the state of the entire system. Monitors and reports can run with any database, including dual event servers, and on any CA Workload Automation AE client computer.

Monitors and reports let you filter and display only the information you are interested in from of a vast collection of data. That is, they are tools that can provide information meaningful to you.

Both monitors and reports filter events by type and by job or groups of jobs. Reports also filter by time (monitors do not filter by time because they provide real-time information).

Note: Because monitors provide a picture of the system's state in real time, they will not provide any information if the scheduler is down. On the other hand, reports provide a picture of the system's state from a historical perspective, not in real time.

Monitors

Monitors provide a real-time view of the system. The following steps are necessary to use a monitor:

1. Define which events to monitor.
2. Run the monitor.

A running monitor is an application that polls the database for new events that meet the selection criteria. Monitors are strictly informational. They provide an up-to-the-minute view of CA Workload Automation AE events as they occur. For box jobs, all job levels can be observed, if appropriate.

Note: Monitor names can only contain the following characters: a-z, A-Z, 0-9, period (.), underscore (_), and hyphen (-). You cannot include spaces or tabs in a monitor name.

Reports

A *report* or browser is a query run against the database, based on the selection criteria defined for that report. Its primary function is to enable you to quickly get specific information, such as the finish time of the database backup for the last two weeks or a list of all jobs that have an alarm associated with them. In addition, all job levels in box jobs can be reported.

Like monitors, a report definition is stored in the database, enabling you to run reports any time, without redefining the criteria. Reports can only display events that are still in the database. Archived events are inaccessible and cannot be displayed.

Note: Report names can only contain the following characters: a-z, A-Z, 0-9, period (.), underscore (_), and hyphen (-). You cannot include spaces or tabs in a report name.

Define a Monitor or Report

To define a monitor or report, issue the jil command and pass it the insert_monbro subcommand followed by a set of attributes.

To define a new monitor or report, you assign it a name and specify attributes that further define its behavior. Using these attributes, you can specify everything from the name to assign to the monitor or report to the events it should track. The monitor or report specification is stored in the database.

Essential Monitor and Report Attributes

The following topics describe the essential monitor and report attributes. You must specify these attributes to define a valid monitor or report.

This section provides the following information for each attribute described:

- Its name.
- Its JIL attribute keyword.
- A description of its use.

Common Essential Attributes—General

The following attributes are required for both monitors and reports. Although defaults might be available, every monitor or report definition must include these attributes, whether by default or by explicit specification.

Monitor or Report Name

`insert_monbro:` *`monbro_name`*

monbro_name

Defines a unique name for the monitor or report.

Limits: A monitor cannot have the same name as a report, but a monitor or report can have the same name as a job.

This value can be up to 30 characters in length and can contain the following characters: A-Z, a-z, 0-9, period (.), underscore (_), and hyphen (-). This value cannot include spaces or tabs.

Mode

`mode:` *`type`*

type

Specifies whether to define a monitor or report.

- Set *type* to **m** or **monitor** to define a monitor.
- Set *type* to **b** or **browser** to define a report.

Common Essential Attributes—Events

The events specification verifies which events to monitor or report. An *event* is any change in the state of a job. Events can be programmatically generated occurrences, such as alarms, or they can be manually generated occurrences, such as starting a job, putting a job on hold, or killing a job.

Monitors and reports can track events by filtering at several levels or based on selected jobs. The events to track are verified by the combination of the various event filters and the job filter, which are specified with any of the essential attributes.

All Events

`all_events:` *`toggle`*

toggle

Specifies whether to track all events for the specified jobs.

- Set *toggle* to **y** or **1** to track all events. In this case, the other event filtering attributes are ignored, and all events, regardless of source, are reported for the selected jobs. These events include job status events and alarms.
- Set *toggle* to **n** or **0** if you do not want to track all events.

Note: Do not define a monitor to monitor all events for all jobs. Instead, use the following command to display the scheduler log in real time:

```
autosyslog -e
```

Running a monitor adds another connection to the database, and establishes a process that continually polls the database. This can significantly impact system performance. Moreover, the scheduler log provides more diagnostic information than a monitor.

Default: 0

Alarms

`alarm:` *`toggle`*

toggle

Specifies whether to track alarms. You can track alarms and job status events in the same monitor or report.

- Set *toggle* to **y** or **1** to track alarms.
- Set *toggle* to **n** or **0** if you do not want to track alarms. This is the default value.

Default: 0

All Job Status Events

`all_status: toggle`

toogle

Specifies whether to track all job status events for the specified jobs. Job status events occur whenever a job's status changes. You can track alarms and job status events in the same monitor or report.

- Set *toggle* to **y** or **1** to track all job status events.
- Set *toggle* to **n** or **0** if you do not want to track all job status events.

Default: 0

Individual Job Status Events

The following table lists the additional monbro attributes used to request the display of individual job status events:

JIL Keyword	Field Name
running	Displays RUNNING status events
success	Displays SUCCESS status events
failure	Displays FAILURE status events
terminated	Displays TERMINATED status events
starting	Displays STARTING status events
restart	Displays RESTART status events

Note: For each JIL keyword, do the following:

- Set toggle to *y* or *1* to track the individual job status event.
- Set toggle to *n* or *0,* if you do not want to track the individual job status events.

Default: *n* or *0*.

Job Filter

`job_filter: type`

type

Defines which jobs to track. Monitors and reports can track events based on selected jobs. The events to track are verified by the combination of the various event filters and the job filter.

- Set *type* to **a** to track all jobs (no job filtering).
- Set *type* to **b** to track a single box and the jobs it contains.
- Set *type* to **j** to track a single job.

If you set *type* to **b** or **j**, you must also define the job_name attribute to specify the name of the box or job to track.

Default: a

Essential Report Attributes

When defining reports, you must specify either the currun or the after_time attribute in addition to the essential attributes described previously. The time criteria specified in these mutually exclusive attributes let you select events that occurred during a particular interval.

Current Run Only

`currun: toggle`

toggle

Specifies whether to report events only for the current or most recent run of the specified jobs.

- Set *toggle* to **y** or **1** to report events only for the current or most recent job run.
- Set *toggle* to **n** or **0** if you do not want to restrict reporting to events for the current or most recent job run. If you set *toggle* to **n** or **0**, you must define the after_time attribute.

This feature is useful for getting a sense of what is happening right now. For example, you could select the job status event RESTART, turn off job filtering, and set this attribute to **y** to see all the jobs that have been automatically restarted in their current or latest run.

Default: 1

Events After a Certain Date and Time

```
after_time: "date_time"
```

date_time

Defines that only events occurring for the specified jobs after the specified date and time are reported.

Default: When you set the currun attribute to **n** or **0**, the after_time attribute defaults to 00:00 (12:00 a.m.) on the current day. When you set the currun attribute to **y** or **1**, the after_time attribute is ignored. When you omit the date from the *date_time* value, after_time defaults to the current day.

Limits: Specify *date_time* in the format "*MM*/*DD*/[*YY*]*YY hh*:*mm*", where *MM* is the month, *DD* is the day, [*YY*]*YY* is the year, *hh* is the hour (in 24-hour format), and *mm* is the minutes. You must enclose the *date_time* value in quotation marks (").

You cannot use the after_time attribute in a monitor definition because monitors only show events as they occur.

Optional Monitor Attributes

The following topics describe optional monitor attributes. You must specify these attributes to define a valid monitor. There are no optional report attributes.

This section provides the following information for each attribute described:

- Its name.
- Its JIL attribute keyword.
- A description of its use.

Verification Required for Alarms

`alarm_verif: *toggle*`

toggle

Specifies whether the monitor should require an operator response to alarm events and provides an accounting of alarm events for which there was an operator response. When the monitor detects an alarm event, it prompts the operator for a response and repeats the prompt every 20 seconds until the operator responds. When the operator responds (typically by entering a comment at the prompt), the monitor time stamps the response and records it in the database, along with the alarm event.

- Set *toggle* to **y** or **1** to require an operator response to alarms.
- Set *toggle* to **n** or **0** if you do not want to require an operator response to alarms.

Default: 0

Define Monitors and Reports Using JIL

Use the following syntax to define a monitor or report using JIL:

```
subcommand:monbro_name
attribute_keyword:value
```

The only difference between defining monitors or reports and defining jobs is that different subcommands are used. Use the following JIL subcommands to define and manage monitors and reports:

- insert_monbro
- update_monbro
- delete_monbro

Example: Define a Monitor Using JIL

This example uses the following JIL statements to define a monitor named Regular. This monitor tracks job status events when a job changes state to RUNNING, SUCCESS, FAILURE, or TERMINATED. It also defines a monitor named Alarm that tracks all alarm events. When the monitor detects an alarm event, it prompts the operator for a response and repeats the prompt every 20 seconds until the operator responds.

```
insert_monbro: Regular
mode: m
running: y
success: y
failure: y
terminated: y

/* Monitor for JUST ALARMS!
* Verification Required is ON so someone must type in a response.
*/

insert_monbro: Alarm
mode: m
alarm: y
alarm_verif: y
```

These JIL statements are stored in the ujo_monbro table in the database.

Example: Define a Report Using JIL

This example uses the following JIL statements to define a report named Alarm_Rep that reports all alarms on any job from June 1, 1997 at 2:00 a.m. to the present.

```
insert_monbro: Alarm_Rep
mode: b
alarm: y
after_time: "06/01/1997 2:00"
```

The quotation marks are required in the after_time value because it contains a colon (:).

Note: Reports can only display events that are still in the database. Archived events are inaccessible and cannot be displayed.

Run a Report or Monitor

To run a report or monitor, run the following command at the command prompt:

```
monbro -N monitor_name
```

Create a Forecast Report

You can generate a report on future job flows based on a date or date range. Forecast reports can help you predict what occurs when a set of conditions are predefined. You can see what happens when values are changed for each forecast period and use this information to plan workflow. The reported job flow displays a list of future jobs based on the dates you specify.

To create a forecast report

1. Do *one* of the following:
   - On UNIX, run the shell that is sourced to use CA Workload Automation AE.

     The UNIX operating system prompt is displayed.
   - On Windows, click Start, Programs, CA, Workload Automation AE, Command Prompt (*instance_name*).

     The CA Workload Automation AE command prompt opens. The command prompt presets all the environment variables for the instance.
2. Enter *one* of the following commands:
   - To report a future job flow for a job:

     ```
     forecast -F "mm/dd/yyyy HH:MM" [-T "mm/dd/yyyy HH:MM"] [-J job_name] [-s] [-h]
     [-n] [-l]
     ```
   - To report a future job flow for a machine:

     ```
     forecast –M machine_name -F "mm/dd/yyyy HH:MM" [-T "mm/dd/yyyy HH:MM"] [-s]
     [-h] [-n] [-l]
     ```
   - To report a future job flow for a job on a specific machine:

     ```
     forecast –M machine_name -F "mm/dd/yyyy HH:MM" [-T "mm/dd/yyyy HH:MM"] [-J
     job_name] [-s] [-h] [-n] [-l]
     ```

   Note: If you omit the -J *job_name* option, the command reports on all jobs.

   The forecast report is created and displayed.

Note: For more information about the forecast command, see the *Reference Guide*.

Example: Create a Forecast Report for a Job

This example generates a forecast report for the testproc job. The report displays when testproc will run between 10:30 a.m. and 10:30 p.m. on December 1, 2010.

```
forecast –J testproc -F "12/01/2010 10:30" -T "12/01/2010 22:30"
```

Example: View the Duration and End Time of a Job

This example generates a forecast report for the test_4 job. The report displays when test_4 will run between 11:40 p.m. on December 14, 2009 and 23:59 hours following that time (the default end time).

```
forecast -F "12/14/2009 23:40" -n
```

The report is similar to the following:

```
Box Name     Job Name     Machine Name     Start Time          Duration     End Time
---         test_4       localhost        12/14/2009 23:40    1             12/14/2009 23:41
```

Appendix A: Legacy Agent Considerations

This section contains the following topics:

Running Jobs on Computers with Legacy CA Workload Automation AE Agents

CA Workload Automation AE can schedule jobs on a computer that is running a previous version (or legacy version) of the CA Workload Automation AE agent. Because legacy agents connect directly to the database in order to add job events, CA Workload Automation AE only works with legacy agents running the same database vendor as the event server. In addition, the CA Workload Automation AE instance identifier (AUTOSERV) must match the instance identifier of the legacy agent.

Define Legacy Agent Computers

The CA Workload Automation AE scheduler can run jobs on both CA Workload Automation AE r11 and r4.5 legacy agent computers. Due to the differences in the communication protocol used by r11 and r4.5 legacy agent computers, the scheduler must know which communication protocol to invoke before contacting the agent computer. This is done by examining the agent's machine definition type attribute. If the type attribute is set to either l or L, the scheduler component prepares the job data using the legacy protocol before sending it to the agent computer.

Before you can run jobs on a legacy agent computer, you must define the computer to CA Workload Automation AE. To define a legacy agent computer, use the following JIL statements:

```
insert_machine: remote_host
type: machine_type
```

remote_*host*

Defines the name of the legacy agent computer.

machine_*type*

Specifies the type of machine you are defining:

l

Indicates a UNIX computer running a CA Workload Automation AE legacy agent. This machine type lets the scheduler know how to use the legacy communication protocol to communicate with the agent and is analogous to an r-type computer for CA Workload Automation AE r11.

L

Indicates a Windows computer running a CA Workload Automation AE legacy agent. This machine type lets the scheduler know how to use the legacy communication protocol to communicate with the agent and is analogous to an n-type computer for CA Workload Automation AE r11.

Notes:

- Legacy agent computers may form part of a virtual machine. The job_load, max_load, and factor attributes continue to support legacy agent computers.
- As part of the event server data migration from a previous product version to CA Workload Automation AE r11, pre-defined machines of type 'r' are converted to type 'l' and pre-defined machines of type 'n' are converted to type 'L'. For more details about data migration from a previous product version, see the *UNIX Implementation Guide* or *Windows Implementation Guide*.

Example: Define a Legacy Agent Computer

This example defines the UNIX computer MYLEGACYUNIXAGENT, which is running a previous version of CA Workload Automation AE:

```
insert_machine: MYLEGACYUNIXAGENT
type: l
```

Define the Legacy Agent Port

In addition to setting the type attribute of the machine definition for each legacy agent computer, the Legacy Agent Port value must also be set in the CA Workload Automation AE environment. This value specifies the port number to be used by the scheduler to communicate with legacy agent computers.

On Windows, the Legacy Agent Port value can be set in the scheduler window of the Administrator utility. On UNIX, you can set this value by locating and updating the AutoRemPort parameter in the $AUTOUSER/config.<instance> file.

Note: Because the CA Workload Automation AE r11 agent has been decoupled from the UNIX internet daemon (inetd), the installation does not add any service entries to either the UNIX services (found in /etc/services) or the inetd configuration files (/etc/inetd.conf).

Change the Database Password for a Unicenter AutoSys JM 4.5 Legacy Agent Computer

To run jobs on a Unicenter AutoSys JM 4.5 legacy agent computer, you must change the database password. The Unicenter AutoSys JM 4.5 legacy agent uses this password to connect to the event server.

To change the database password for a Unicenter AutoSys JM 4.5 legacy agent computer

1. Log on to CA Workload Automation AE as the EDIT superuser and enter the following command at the UNIX operating system prompt or the Windows instance command prompt:

   ```
   autosys_secure
   ```

   The following menu appears:

   ```
   Please select from the following options:
   [1] Activate EEM instance security.
   [2] Manage EDIT/EXEC superusers.
   [3] Change database password.
   [4] Change remote authentication method.
   [5] Manage user@host users.
   [6] Get Encrypted Password.
   [0] Exit CA WAAE Security Utility.
   ```

2. Enter 3 and press the Enter key.
3. Enter the password information when prompted.

   The password is changed. The following message appears:

   ```
   CAUAJM_I_60059 Password change successful.
   ```

How Jobs Are Run On Legacy Agent Computers

The process by which CA Workload Automation AE can run jobs directly on a legacy agent computer is as follows:

- After evaluating the job start conditions, the scheduler places the job in a STARTING status.
- The scheduler recognizes the type of the machine as a computer running a legacy agent.
- The scheduler obtains the port number of the legacy agent computer from its configuration settings.
- The scheduler initiates communication with the legacy agent computer. On Windows, the agent service starts the agent process. On UNIX, the internet daemon (inetd) starts the agent process.
- The scheduler prepares the job details using the legacy agent communication protocol. The scheduler then sends the information to the newly started agent process.
- The legacy agent completes the communication protocol with the scheduler and starts the job.
- The legacy agent puts a RUNNING event directly into the event server.
- The scheduler processes the RUNNING event.
- After the job completes, the legacy agent puts a SUCCESS, FAILURE, or TERMINATED event directly into the event server based on the exit code of the job.
- The scheduler processes the end status event.

Notes:

- If the scheduler log reports an error while trying to run a job on a computer with a legacy Agent, see the Agent log file on the remote computer for details.
- The legacy agent log may report some database errors while trying to send job status events even when the scheduler log shows that the job has completed successfully. The errors are due to the differences between the new database tables and the tables expected by the legacy agent. These database errors do not prevent the job event from being sent to the event server and must be ignored.

Index