

CA Workload Control Center

Workload Scheduling Guide

r11.3



This documentation and any related computer software help programs (hereinafter referred to as the "Documentation") are for your informational purposes only and are subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be used or disclosed by you except as may be permitted in a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2010 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

| | |
|--|-----------|
| Chapter 1: Introduction | 11 |
| Intended Audience | 11 |
| Quick Edit and Application Editor | 11 |
| Agents and Agent Plug-ins | 12 |
| Related Documentation | 13 |
| | |
| Chapter 2: Working with Calendars | 15 |
| Calendars | 15 |
| Cycles | 15 |
| Create a Standard Calendar | 16 |
| Create a Cycle | 17 |
| Create an Extended Calendar | 19 |
| Extended Calendar Properties | 21 |
| | |
| Chapter 3: Working with Jobs and Job Flows | 23 |
| Jobs | 23 |
| Scheduling Jobs and Job Dependencies | 24 |
| How to Create a Job Flow in Application Editor | 25 |
| Add a Job to a Job Flow Using the Palette | 26 |
| Import Jobs | 26 |
| Specify Job Properties in Application Editor | 28 |
| Add a Global Variable to a Job Flow | 28 |
| Add a Logical Operator to a Job Flow | 30 |
| Draw Dependency Links | 31 |
| Set a Value for a Job Dependency | 32 |
| Set a Value for a Global Variable Dependency | 33 |
| Commit Changes in Application Editor | 34 |
| Create a Job in Quick Edit | 35 |
| Create a Global Variable in Quick Edit | 36 |
| Commit Changes in Quick Edit | 37 |
| | |
| Chapter 4: Common Job Properties | 39 |
| Specifying the Job Owner | 39 |
| Define Starting Conditions for a Job | 40 |
| Job Status Dependencies | 41 |
| Cross-Instance Job Dependencies | 42 |

| | |
|---|----|
| Exit Code Dependencies | 42 |
| Look-Back Dependencies | 43 |
| Global Variable Dependencies | 44 |
| Starting Condition Examples | 45 |
| Primary Properties | 47 |
| Define Resource Dependencies for a Job | 50 |
| Resource Dependency Examples | 52 |
| Defining Date and Time Dependencies | 53 |
| Time Zone | 54 |
| Custom Calendars | 54 |
| Must Start Times and Must Complete Times | 55 |
| Date and Time Dependency Examples | 58 |
| Schedule Properties | 60 |
| Specifying Environment Variables | 63 |
| Environment Variables Examples | 63 |
| Job Profile Examples | 66 |
| Environment Properties | 67 |
| Defining Job Termination Conditions | 70 |
| Exit Codes to Indicate Job Failure or Success | 70 |
| Maximum Runtime | 72 |
| Automatic Deletion When a Job Completes | 72 |
| Termination of a Failed Box Job or Containing Box | 73 |
| Termination Properties | 73 |
| Setting Up Notifications | 75 |
| Specify to Send an Alarm Under Certain Conditions | 76 |
| Specify to Send a Notification When a Job Completes | 77 |
| Specify to Create a Service Desk Request When a Job Fails | 78 |
| Notification Properties | 79 |
| Specify the Users with Edit and Execute Permissions | 82 |
| Native Security Permissions Properties | 83 |

Chapter 5: Box Jobs 85

| | |
|-------------------------------|----|
| Box Jobs | 85 |
| Define a Box Job | 86 |
| Box Job Properties | 87 |
| Grouping Jobs and Boxes | 88 |
| Add a Job to a Box Job | 89 |

Chapter 6: Command Jobs 91

| | |
|----------------------------|----|
| Command Jobs | 91 |
| Define a Command Job | 92 |

| | |
|--|------------|
| Command Job Examples (UNIX) | 93 |
| Command Job Examples (Windows) | 96 |
| Command Properties | 98 |
| Verify File Space Before a Job Starts..... | 102 |
| Pass Positional Arguments in a Command Job | 103 |
| UNIX Environment Variables | 105 |
| Define Alternative Error, Input, and Output Sources and Destinations | 106 |
| | |
| Chapter 7: User Defined Jobs | 109 |
| User Defined Jobs | 109 |
| Define a User Defined Job | 110 |
| User Defined Properties | 110 |
| | |
| Chapter 8: File Watcher Jobs | 115 |
| File Watcher Jobs | 115 |
| Define a File Watcher Job | 116 |
| File Watcher Job Examples | 117 |
| File Watcher Properties | 117 |
| | |
| Chapter 9: Entity Bean and Session Bean Jobs | 119 |
| Entity Bean Jobs | 119 |
| Session Bean Jobs | 120 |
| Define an Entity Bean Job | 122 |
| Entity Bean Job Examples | 123 |
| Entity Bean Properties | 125 |
| Define a Session Bean Job | 129 |
| Session Bean Job Examples | 130 |
| Session Bean Properties | 131 |
| Edit Parameters | 134 |
| | |
| Chapter 10: JMS Publish and JMS Subscribe Jobs | 137 |
| JMS Publish and JMS Subscribe Jobs | 137 |
| Define a JMS Publish Job | 141 |
| JMS Publish Job Examples | 142 |
| JMS Publish Properties | 143 |
| Define a JMS Subscribe Job | 146 |
| JMS Subscribe Job Examples | 147 |
| JMS Subscribe Properties | 147 |

| | |
|--|------------|
| Chapter 11: JMX Jobs | 151 |
| JMX Jobs | 151 |
| Payload Producing Jobs | 152 |
| Define a JMX-MBean Attribute Get Job | 154 |
| JMX-MBean Attribute Get Job Examples | 155 |
| JMX-MBean Attribute Get Properties | 155 |
| Define a JMX-MBean Attribute Set Job | 157 |
| JMX-MBean Attribute Set Job Examples | 158 |
| JMX-MBean Attribute Set Properties | 158 |
| Define a JMX-MBean Create Instance Job | 161 |
| JMX-MBean Create Instance Job Examples | 162 |
| JMX-MBean Create Instance Properties | 162 |
| Define a JMX-MBean Operation Job | 164 |
| JMX-MBean Operation Job Examples | 165 |
| JMX-MBean Operation Properties | 165 |
| Define a JMX-MBean Remove Instance Job | 168 |
| JMX-MBean Remove Instance Job Examples | 169 |
| JMX-MBean Remove Instance Properties | 169 |
| Define a JMX-MBean Subscribe Job | 171 |
| JMX-MBean Subscribe Job Examples | 172 |
| JMX-MBean Subscribe Properties | 173 |
| Chapter 12: HTTP, POJO, RMI, and Web Service Jobs | 177 |
| HTTP Jobs | 177 |
| POJO Jobs | 178 |
| RMI Jobs | 179 |
| Web Service Jobs | 180 |
| Define an HTTP Job | 181 |
| HTTP Job Examples | 182 |
| HTTP Properties | 183 |
| Define a POJO Job | 188 |
| POJO Job Examples | 189 |
| POJO Properties | 189 |
| Define an RMI Job | 191 |
| RMI Job Example | 192 |
| RMI Properties | 192 |
| Define a Web Service Job Using Manual Entries | 194 |
| Define a Web Service Job Using the Wizard | 195 |
| Web Service Job Examples | 196 |
| Web Service Properties | 198 |

| | |
|---|------------|
| Chapter 13: Database Jobs | 203 |
| Database Jobs | 203 |
| How Database Trigger Jobs Differ from Database Monitor Jobs | 204 |
| Define a Database Monitor Job | 205 |
| Database Monitor Job Examples | 206 |
| Database Monitor Properties | 207 |
| Define a Database Stored Procedure Job | 209 |
| Database Stored Procedure Job Examples | 210 |
| Database Stored Procedure Properties | 214 |
| Supported Data Types | 217 |
| Define a Database Trigger Job | 219 |
| Database Trigger Job Examples for Oracle | 220 |
| Database Trigger Job Examples for Microsoft SQL Server | 224 |
| Database Trigger Job Examples for IBM DB2 | 227 |
| Database Trigger Properties | 228 |
| Define an SQL Job | 231 |
| SQL Job Examples for Oracle | 232 |
| SQL Job Examples for Microsoft SQL Server | 234 |
| SQL Job Examples for IBM DB2 | 236 |
| SQL Properties | 238 |
| Chapter 14: Monitoring Jobs | 241 |
| Monitoring Jobs | 241 |
| File Trigger Jobs | 242 |
| Windows Event Log Monitoring Jobs | 243 |
| Define a CPU Monitoring Job | 244 |
| CPU Monitoring Job Examples | 248 |
| CPU Monitoring Properties | 248 |
| Define a Disk Monitoring Job | 252 |
| Disk Monitoring Job Examples | 253 |
| Disk Monitoring Properties | 257 |
| Define a File Trigger Job | 261 |
| File Trigger Job Examples | 262 |
| File Trigger Properties | 266 |
| Define an IP Monitoring Job | 271 |
| IP Monitoring Job Examples | 272 |
| IP Monitoring Properties | 272 |
| Define a Process Monitoring Job | 274 |
| Process Monitoring Job Examples | 275 |
| Process Monitoring Properties | 277 |
| Define a Text File Reading and Monitoring Job | 279 |

| | |
|---|------------|
| Text File Reading and Monitoring Job Examples | 280 |
| Text File Reading and Monitoring Properties | 282 |
| Define a Windows Event Log Monitoring Job | 286 |
| Windows Event Log Monitoring Job Examples | 287 |
| Windows Event Log Monitoring Properties | 288 |
| Define a Windows Service Monitoring Job | 292 |
| Windows Service Monitoring Job Examples | 293 |
| Windows Service Monitoring Properties | 293 |
| Chapter 15: Oracle E-Business Suite Jobs | 297 |
| Oracle E-Business Suite Jobs | 297 |
| Define an Oracle E-Business Suite Copy Single Request Job | 298 |
| Oracle E-Business Suite Copy Single Request Job Example..... | 299 |
| Oracle E-Business Suite Copy Single Request Properties | 300 |
| Define an Oracle E-Business Suite Request Set Job | 301 |
| Specify Data for an Individual Program in a Request Set..... | 303 |
| Oracle E-Business Suite Request Set Job Example | 306 |
| Oracle E-Business Suite Request Set Properties | 306 |
| Define an Oracle Applications Single Request Job | 310 |
| Oracle E-Business Suite Single Request Job Example | 311 |
| Oracle E-Business Suite Single Request Properties | 312 |
| Chapter 16: SAP Jobs | 317 |
| SAP Jobs | 318 |
| How to Use Filters to Define SAP Jobs | 319 |
| Create an SAP Filter | 320 |
| Display a List of Jobs on the SAP System | 322 |
| Copy a Job from the SAP System to a Job Flow | 323 |
| Define an SAP Batch Input Session Job | 324 |
| SAP Batch Input Session Job Example | 325 |
| SAP Batch Input Session Properties..... | 325 |
| Define an SAP Business Warehouse InfoPackage Job | 329 |
| SAP BW InfoPackage Job Examples | 330 |
| SAP BW InfoPackage Properties | 330 |
| Define an SAP Business Warehouse Process Chain Job | 333 |
| SAP BW Process Chain Job Example | 334 |
| SAP BW Process Chain Properties | 334 |
| Define an SAP Data Archiving Job | 336 |
| SAP Data Archiving Job Example | 337 |
| SAP Data Archiving Properties | 338 |
| Archiving Parameters | 339 |

| | |
|--|----------------|
| Print Parameters | 342 |
| Define an SAP Event Monitor Job | 346 |
| SAP Event Monitor Job Example | 347 |
| SAP Event Monitor Properties | 348 |
| Define an SAP Job Copy Job | 350 |
| SAP Job Copy Job Example | 351 |
| SAP Job Copy Properties | 351 |
| Define an SAP Process Monitor Job | 354 |
| SAP Process Monitor Job Example | 355 |
| Using Success and Failure Messages within an SAP Job | 356 |
| SAP Process Monitor Properties | 357 |
| Define an SAP R/3 Job | 360 |
| SAP R/3 Job Examples | 361 |
| SAP R/3 Properties | 363 |
| Define Recipients for the Job's Output | 367 |
| Spool List Recipient Properties | 369 |
| Define Print or Archive Parameters for a Step | 370 |
| Step Parameters - Command Properties | 372 |
| Step Parameters - Output Properties | 374 |
| Step parameters - Archive Properties | 377 |
| Send the SAP Spool File by Email on Step Completion or Failure | 380 |
| Chapter 17: PeopleSoft Jobs | 381 |
| PeopleSoft Jobs | 381 |
| Define a PeopleSoft Job | 382 |
| Mapping of PeopleSoft Field Names to PeopleSoft Job Properties | 383 |
| PeopleSoft Job Examples | 384 |
| PeopleSoft Properties | 388 |
| Chapter 18: FTP and Secure Copy Jobs | 397 |
| FTP Jobs | 397 |
| Secure Copy Jobs | 397 |
| Define a File Transfer Protocol (FTP) Job | 398 |
| FTP Job Examples | 399 |
| File Transfer Protocol Properties | 401 |
| Define a Secure Copy Job | 405 |
| Secure Copy Job Examples | 406 |
| Secure Copy Properties | 407 |
| Chapter 19: i5/OS Jobs | 411 |
| i5/OS Jobs | 411 |

| | |
|--|----------------|
| Running UNIX Workload on a System i5 Computer | 412 |
| i5/OS Naming Conventions | 412 |
| Define an i5/OS Job..... | 413 |
| i5/OS Job Examples | 414 |
| i5/OS Properties | 418 |
| Pass Positional Parameters | 423 |
| Use a User's Library List | 423 |
| Pass Keyword Parameters to SBMJOB..... | 424 |
| Returning a Job's Exit Status to CA Workload Automation AE | 425 |
| Send a Program's Return Code | 425 |
| Send a User-defined Exit Code | 426 |
| Specify Data for a Local Data Area..... | 427 |
| Chapter 20: z/OS Jobs | 429 |
| z/OS Jobs | 429 |
| z/OS Data Set Trigger Jobs..... | 430 |
| Define a z/OS Data Set Trigger Job | 431 |
| z/OS Data Set Trigger Job Examples | 432 |
| z/OS Data Set Trigger Properties | 436 |
| Define a z/OS Manual Job | 439 |
| z/OS Manual Job Examples | 440 |
| z/OS Manual Properties | 440 |
| Define a z/OS Regular Job | 441 |
| z/OS Regular Job Examples | 442 |
| z/OS Regular Job Properties | 442 |
| Index | 445 |

Chapter 1: Introduction

This section contains the following topics:

- [Intended Audience](#) (see page 11)
- [Quick Edit and Application Editor](#) (see page 11)
- [Agents and Agent Plug-ins](#) (see page 12)
- [Related Documentation](#) (see page 13)

Intended Audience

This guide is intended for schedulers, supervisors, application developers and users, console operators—anyone who is responsible for defining and scheduling workload using CA WCC. This guide applies to the Application Editor, Quick Edit, and Quick View components of CA WCC, all of which show workload object properties; however, the tasks and examples apply to Application Editor and Quick Edit only because they relate to scheduling, not monitoring. Refer to the related documentation for other CA WCC components.

Quick Edit and Application Editor

Application Editor lets you work with a graphical representation of jobs in a job flow. The job flow shows the relationships between jobs and their dependencies. You can manage individual jobs using Application Editor; however, the focus is on the job flow.

Quick Edit lets you create, edit, delete, and search for individual objects: jobs, calendars, and global variables. In Quick Edit, you can view a job flow but you cannot make any changes to it.

Agents and Agent Plug-ins

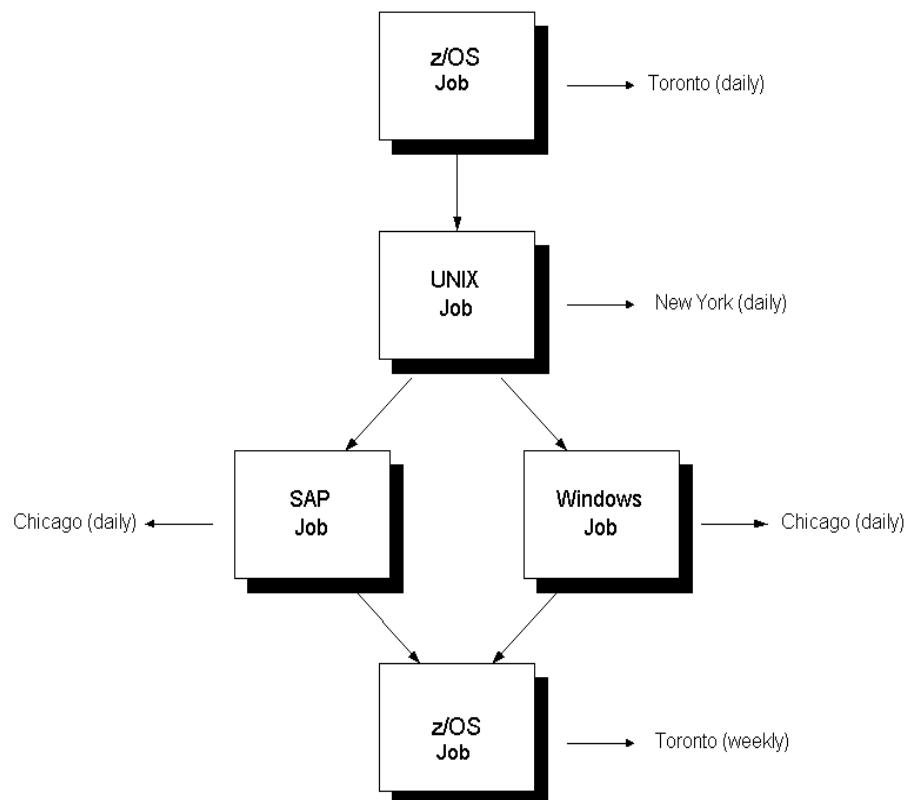
Agents are the key integration components of CA workload automation products. Agents let you automate, monitor, and manage workload on all major platforms, applications, and databases. To run workload on a particular system, you install an agent on that system. If your workload must run on a UNIX computer, 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:



Related Documentation

In addition to this guide, CA WCC provides online help for each component of the interface, and other role-based guides. The help is available from the Help link within each CA WCC component and from the bookshelf included on the product DVD. The guides are accessible in PDF format from the bookshelf included on the product DVD.

For complete information about your scheduling manager, see the CA Workload Automation AE documentation.

Chapter 2: Working with Calendars

This section contains the following topics:

- [Calendars](#) (see page 15)
- [Cycles](#) (see page 15)
- [Create a Standard Calendar](#) (see page 16)
- [Create a Cycle](#) (see page 17)
- [Create an Extended Calendar](#) (see page 19)

Calendars

Calendars are used for scheduling jobs to identify the days when jobs should or should not process. You can define a custom calendar to include any number of dates. You can then associate any calendar with one or more jobs. A custom calendar can be created using a standard calendar or an extended calendar.

A *standard calendar* is a set of manually selected, user-specified dates. There are two types of standard calendars: run and exclude. A *run calendar* specifies the days on which a job should run, such as the first weekday of every month. Conversely, an *exclude calendar* specifies the days on which a job should not run. For example, to help ensure that a job never runs on any corporate holiday, you could create a calendar named Holidays that has each corporate holiday selected.

An *extended calendar* is a dynamic calendar that is generated automatically based on a combination of holiday calendars, cycles, and conditions defined.

Each calendar is stored in the database as a separate object with a unique name.

Cycles

Cycles are used as processing periods in an extended calendar definition. These user-defined date ranges make it easy to define rules for non-standard processing periods, such as business cycles, quarters, and accounting periods. Cycles can be set to repeat every year.

Create a Standard Calendar

You can create a standard calendar to define specified dates on which jobs should run.

To create a standard calendar

1. Open Quick Edit.
2. Click Create.
The Create Object dialog opens.
3. Select the server where you want to store the calendar from the Server drop-down list.
4. Click the Calendar link.
The Calendar Properties section appears.
5. Specify the following required property:

Name

Specifies a name for the calendar.

Limits: Up to 30 characters

6. (Optional) Specify when the calendar starts.
 - a. Select the starting month.
 - b. Select the starting year.
 - c. Click Go.
7. Do **one** of the following:
 - Click individual days in the calendar to set the processing days.
 - Define rules in the Generate Days section for determining processing and blocked days in the calendar and for rescheduling processing when schedule conflicts arise.
8. Click Commit.

The standard calendar is created.

More information:

[Calendars](#) (see page 15)

Create a Cycle

You can create a cycle to define date ranges to be used as input to an extended calendar.

To create a cycle definition

1. Open Quick Edit.
2. Click Create.
The Create Object dialog opens.
3. Select the server where you want to store the cycle from the Server drop-down list.
4. Click the Cycle link.
The Cycle Properties section appears.
5. Specify the following required properties:

Name

Specifies the name for a cycle.

Limits: Up to 30 characters

6. (Optional) Select the Repeat every year check box to repeat the cycle every year.
7. (Optional) Add one or more periods to the cycle, as follows:
 - a. Select Add in the Periods field.
 - b. Click Go.
 - c. Specify a Start and End date for the period. You can click  to select the period dates if the Repeat every year check box is not selected.
- Note:** You must add at least one period, but you cannot exceed 30 periods per cycle.
8. Click Commit.

The cycle is created.

Example: Create a cycle and associate it with an extended calendar

To associate a cycle with an extended calendar, you first create the cycle with the periods defined. Then, when you create the extended calendar with which you want to associate it, enter CYCLE in the Conditions field and select the correct cycle from the Cycle drop-down list in the extended calendar. You can click Generate preview to view the processing dates selected.

More information:

[Cycles](#) (see page 15)

[Create an Extended Calendar](#) (see page 19)

Create an Extended Calendar

You can create an extended calendar to define processing days, using a combination of a holiday calendar, cycle, and conditions, for scheduled jobs.

To create an extended calendar

1. Open Quick Edit.
2. Click Create.
The Create Object dialog opens.
3. Select the server where you want to store the calendar from the Server drop-down list.
4. Click the Extended Calendar link.
The Extended Calendar Properties section appears.
5. Specify the following required properties:

Name

Specifies the name for an extended calendar.

Limits: Up to 30 characters

Adjustment

Specifies an integer to adjust the days generated by the other criteria to indicate a specified number of days before or after the actual generated day.

6. (Optional) Specify any of the following properties in the Primary section:

Conditions

Specifies date conditions for the extended calendar.

Note: Clicking  opens an Edit Text dialog box where you can specify the date conditions.

Examples:

- DAILY - Selects all dates after the current date
- MON - Selects all Mondays after the current date
- FOMWEEK - Selects the first weekday of each month after the current date
- EOMWEEK - Selects the last weekday of each month after the current date
- XFOM - Selects all days except the first day of each month after the current date

Cycle name

Specifies the name of an existing cycle.

Note: Clicking  opens the Search Cycle dialog where you can search for a cycle.

7. (Optional) Specify the following properties in the Holiday Action section:

Reschedule rule

Specifies a rule for holidays, as follows:

- Do not schedule the job on holidays
- Schedule only on a holiday
- Schedule anyway, ignoring holidays
- Schedule on the next day
- Schedule on the next workday
- Schedule on the previous workday

Calendar

Specifies the name of a standard calendar that defines the holidays.

Note: Clicking  opens the Search Calendar dialog where you can search for a calendar.

8. (Optional) Specify any of the following properties in the Non Workday Action section:

Non workday action

Specifies a rule for non-workdays, as follows:

- Schedule anyway, ignoring workdays
- Schedule only on non-workdays
- Schedule only on workdays
- Schedule on the next workday
- Schedule on the previous workday

Workdays

Specifies the workdays.

9. (Optional) Click Generate Preview to open the Extended Calendar Preview dialog so you can preview the dates generated by the rules in the extended calendar.

10. Click Commit.

The extended calendar is created.

Note: If you create an extended calendar by entering a value in the Name field only and saving it, all days are selected from the current day forward, for a total of 366 days.

Example: Create an extended calendar based on field adjustment

To create an extended calendar that processes associated jobs one day after each holiday, you would first create a calendar defining the holidays for the year, then create an extended calendar with the following options selected:

- Your holidays calendar in the Calendar field
- Schedule only on a holiday in the Non Workday Action list
- 1 in the Adjustment field

More information:

[Create a Cycle \(see page 17\)](#)

Extended Calendar Properties

The Primary, Holiday Action, and Non Workday Action categories for extended calendars include the following properties:

Adjustment

Defines the number of days before (negative number) or after (positive number) the other properties to run an associated job. Use adjustments if the other properties do not define the necessary days.

Limits: -100 - 100, including 0

Example: To run a job the day after every holiday, enter "holidays" in the Conditions field and specify +1 in the Adjustment field.

JIL attribute: adjust

Calendar

(Optional) Defines the calendar name to use for holiday dates.

Limits: Up to 30 characters

JIL attribute: holcal

Conditions

(Optional) Defines when to use the holiday calendar and cycle.

Limits: Up to 255 characters

Example: DAILY uses the calendar or cycle every day

Note: For more information, see the *CA Workload Automation AE Reference Guide*.

JIL attribute: condition

Cycle name

(Optional) Defines the cycle name to use and is only valid when the cycle keywords appear in the Conditions field.

Limits: Up to 30 characters

JIL attribute: cyccal

Name

Specifies the extended calendar name.

Limits: Up to 30 characters

JIL attribute: extended_calendar

Non Workday Action

(Optional) Indicates the action to take when the selected day falls on a non-workday, as follows:

- Schedule anyway, ignoring workdays
- Schedule only on non-workdays
- Schedule only on workdays
- Schedule on the next workday
- Schedule on the previous workday

JIL attribute: non_workday

Reschedule rule

(Optional) Indicates the action to take if the selected day is a holiday, as follows:

- Do not schedule the job on holidays
- Schedule only on a holiday
- Schedule anyway, ignoring holidays
- Schedule on the next day
- Schedule on the next workday
- Schedule on the previous workday

JIL attribute: holiday

Workdays

(Optional) Specifies whether the days of the week are workdays.

JIL attribute: workday

Chapter 3: Working with Jobs and Job Flows

This section contains the following topics:

- [Jobs](#) (see page 23)
- [Scheduling Jobs and Job Dependencies](#) (see page 24)
- [How to Create a Job Flow in Application Editor](#) (see page 25)
- [Create a Job in Quick Edit](#) (see page 35)
- [Create a Global Variable in Quick Edit](#) (see page 36)
- [Commit Changes in Quick Edit](#) (see page 37)

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 that specify the job's properties and behavior. For example, you can specify conditions that determine when and where a job runs.

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 file, an FTP job to download a file from a server, or an SAP Event Monitor 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 with a User Defined job.

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

You can define jobs using the Application Editor or Quick Edit CA WCC components. Both components set the same properties and the job definition is always stored in the database. You can also modify and delete existing job definitions.

More information:

- [Common Job Properties](#) (see page 39)
- [Quick Edit and Application Editor](#) (see page 11)

Scheduling Jobs and Job Dependencies

CA WCC lets you interactively set the properties that describe when, where, and how a job should run. In addition, CA WCC lets you define calendars, global variables, and jobsets, and monitor and manage jobs. The fields in CA WCC correspond to CA Workload Automation AE JIL subcommands and attributes.

You can define job properties that

- Specify starting conditions for jobs within your job definitions.
- Set date and time dependencies for a job so that it runs automatically on specific days and at specific times.
- Base job-starting conditions on a calendar event (basic time/day or custom calendar), job dependency status, or file arrival.
- Specify resource dependencies so that a job will not start execution until the required resources are available.

Additionally, you can specify job termination conditions within your job definitions. For example, you can specify exit codes that determine the success or failure of a job or the number of minutes to wait before terminating the job.

How to Create a Job Flow in Application Editor

A job flow is a graphical representation of the relationships between CA WCC jobs and their dependencies. The job flow displays the conditions of the relationships that must be met before a job can run, and the successor jobs are triggered when a job finishes running.

To create a job flow, follow these steps:

1. Open Application Editor.
2. Add jobs to the job flow using one of these methods:
 - [Add a job using the Palette](#) (see page 26).
 - [Import jobs](#) (see page 26).
3. [Specify job properties for each job](#) (see page 28).
4. (Optional) Add one or both of the following dependent objects:
 - [Add global variables](#) (see page 28).
 - [Add logical operators](#) (see page 30).
5. [Draw dependency links](#) (see page 31).
6. Set values for dependency links, as required:
 - [Set a value for a job dependency](#) (see page 32)
 - [Set a value for a global variable dependency](#) (see page 33)
7. [Commit changes in Application Editor](#) (see page 34).

Add a Job to a Job Flow Using the Palette

The Palette in Application Editor provides icons for all the available job types.

To add a job to a job flow using the Palette

1. Open Application Editor.

A blank graph appears in the workspace.

Note: You can open the last graph you worked on by selecting Load workspace from the Select an action menu.

2. Locate the job you want to add from the Palette.

Note: You can change which jobs appear in the Palette using the Filter Job Types tab of the Customize dialog.

3. Drag and drop the job icon onto the workspace.

The job icon appears on the workspace.

4. Repeat Steps 2 and 3 for each job you want to add to the job flow.

Notes:

- If a job must run after another job, place it below that job on the workspace.
- If a job must run before another job, place it above the job on the workspace.
- If a job must run at the same time as another job, place it beside that job on the workspace.

The jobs are added to the job flow.

5. Select Save workspace from the list of actions and click Go.

The workspace is saved.

Note: The job is not created on the CA Workload Automation AE server until you commit the workspace.

Import Jobs

You can import one or more jobs into Application Editor to view the flow, add jobs, edit job properties, add dependencies, or modify existing dependencies.

To import jobs

1. Open Application Editor.
2. Select Import from scheduler from the drop-down list in the Flow section and click Go.

The Import Jobs dialog opens.

3. Select a server from the Server drop-down list.
4. Specify one or more of the following and click Go:
 - Name
 - Group
 - Application

Notes:

- The following are permitted:
 - Valid job name, group, or application—Finds the specified jobs on the selected server that match the specified criteria.
 - Asterisk (*), percent (%), or question mark (?), used as a wildcard with a partial name, where the * or % represents zero or more characters, or ? represents a single character—Finds all jobs on the selected server that contain the partial job, group, or application name. For example, entering n* in the Name field only would find all jobs on the server that begin with the letter n.
 - Asterisk (*) or percent (%)—Finds all jobs on the selected server when the wildcard is specified for the Name field and the other two fields are blank.
- With the Name field blank, specifying an asterisk (**) or percent (%%) in the Group or Application field will return a list of all jobs that have the Group or Application attribute specified.
- With the Name field blank, specifying an asterisk (*) or percent (%) in the Group and Application fields finds all jobs on the selected server.

A list of jobs appears in the Search Results section.

Note: The Search Results section may include jobs and Box jobs.

5. Do one of the following:
 - Select the check box for the jobs and box jobs you want to import, and click Import Selected.
 - Click Import All.

The Import Jobs dialog closes, and the specified jobs, box jobs, and their dependencies appear in the Graph.

6. Select Save workspace from the drop-down list and click Go.

The workspace is saved.

Specify Job Properties in Application Editor

You specify properties to define a job. Each job has required properties, which are identified with a red dot.

Note: For specific job details, refer to the related job chapter in this guide.

To specify job properties in Application Editor

1. Open the job flow in Application Editor.
2. Right-click the job icon in the Graph and select Edit.
The properties section for the job opens in the lower half of Application Editor.
3. Complete the required properties.
4. Complete other optional properties as required.
5. Click Apply in the Properties section.
The job properties are saved to the Application Editor workspace and the Properties section closes.
6. Select Save workspace from the list of actions and click Go.
The workspace is saved.

More information:

[Common Job Properties](#) (see page 39)

Add a Global Variable to a Job Flow

You can add a global variable to a job flow if you have a set of jobs you want to run based upon specific criteria.

When you add a global variable to a job flow in Application Editor, it is actually a global variable condition, not a global variable object. That is, the global variable is part of the dependency string, and processing will proceed when the variable is set with that value. These global variables do not necessarily exist on the CA Workload Automation AE server before being set to satisfy the condition. Adding a global variable in the job flow automatically meets its requirement and triggers its dependent jobs.

Notes:

- Global variables you add to the flow do not have to be defined in the CA Workload Automation AE database to be used in the flow.
- You cannot assign a value to a global variable that you add to the flow; however, you can define the properties of the dependency link that connects the global variable to another object. The dependency link defines the value that the global variable must be equal to, not equal to, less than, or greater than for the dependency condition to be met.
- The global variable you add to the flow is included in the dependency statement of the job that depends on it.

You can set a global variable value in a sendevent command. CA Workload Automation AE keeps it in memory so it can be used.

To add a global variable to a job flow

1. Open the job flow in Application Editor.
2. Drag the Global Variable icon from the Palette and drop it onto the workspace.
The Global Variable icon appears on the workspace.
3. Right-click the Global Variable icon and select Edit.
The Edit Global Variable dialog opens.
4. Specify the following global variable property:

Name

Specifies the name of the global variable.

Limits: Up to 64 characters.

5. Click OK.
The name of the global variable changes in the graph.
6. Select Save workspace from the list of actions and click Go.
The global variable is added and the workspace is saved.

Add a Logical Operator to a Job Flow

A logical operator uses the logically-combined result of two or more objects in the job flow to control subsequent processing. Objects are connected to logical operators using dependency links. You must set the value of the dependency link, such as the result of a job run from objects that precede the logical operator. The dependency link from a logical operator to the next object in the job flow does not have a value; the operator itself controls subsequent processing.

The following logical operators are available:

AND

Subsequent processing occurs as the result of the logical AND of two or more preceding objects. For example, if you want Job-C to run when Job-A and Job-B both complete successfully, the dependency statement for Job-C is: $s(Job-A) \& s(Job-B)$.

OR

Subsequent processing occurs as the result of the logical OR of two or more preceding objects. For example, if you want Job-C to run when either Job-A or Job-B completes successfully, the dependency statement for Job-C is: $s(Job-A) | s(Job-B)$.

To add a logical operator to a job flow

1. Open the job flow in Application Editor.
2. Drag the Logical Operator icon from the Palette and drop it onto the workspace.
The Logical Operator icon appears on the workspace.
3. Select Save workspace from the list of actions and click Go.
The workspace is saved.

Draw Dependency Links

You can create dependencies between jobs, global variables, and logical operators by drawing dependency links in the job flow. A dependency is created by linking two objects in the job flow. The dependency link appears as an arrow pointing from the dependent object to the object that depends on it. The dependency link property defines the condition of the object that must be met before the next step in the job flow is performed. You can select the properties for a dependency link originating from a job or a global variable by right-clicking the link and selecting Edit.

To draw dependency links

1. Open the job flow in Application Editor.
2. Click  on the Graph toolbar.
3. Hold down the mouse key and draw a line between two objects on the graph.

Notes:

- To set up a dependency between two jobs, select a predecessor job and drag the mouse to the next job to be released (successor).
- To set up a dependency between a global variable and a job or logical operator, draw the line starting from the global variable.

A dependency line appears between the two objects.

4. Repeat Step 3 to draw all the dependency links.
5. Click  to organize the layout of the job flow.
The dependency links are drawn on the graph in the workspace.
6. Select Save workspace from the list of actions, and click Go.
The workspace is saved.

More information:

[Create a Global Variable in Quick Edit](#) (see page 36)

Set a Value for a Job Dependency

You set a value for a dependency link from a job in the job flow. The value sets the condition that must be met before the next step in the job flow is performed. For example, if JOB_A runs successfully, processing will continue.

To set a value for a job dependency

1. Open the job flow in Application Editor.
2. Right-click the dependency link from a job in the job flow and select Edit.
The Edit Job Dependency dialog opens.
3. Complete the following fields:

Dependency type

Specifies the job status for the dependent job, as follows:

- SUCCESS—If the job completes successfully, processing continues with the next step in the flow. The link color is green.
- FAILURE—If the job fails, processing continues with the next step in the flow. The link color is red.
- DONE—If the job indicates that it is done, processing continues with the next step in the flow. The link color is blue.
- TERMINATED—If the job is terminated, processing continues with the next step in the flow. The link color is magenta.
- NOTRUNNING—If the job is not running, processing continues with the next step in the flow. The link color is gray.
- EXITCODE—If the job returns the exit code you select, processing continues with the next step in the flow. The link color is black.

[Operator]

Specifies the operator to use when comparing the specified value with the exit code (=, !=, >, <).

Note: This field is only active when EXITCODE is selected for the dependency type.

Value

Specifies the value to which the exit code is compared.

Note: This field is only active when EXITCODE is selected for the dependency type.

4. Click OK.

The job dependency value is set and the color of the dependency link changes to match the dependency type.

Set a Value for a Global Variable Dependency

You set a value for a dependency link from a global variable in the job flow. The value sets the condition that must be met before the next step in the job flow is performed. For example, if the global variable run_count is equal to 5, processing will continue.

To set a value for a global variable dependency

1. Open the job flow in Application Editor.
2. Right-click the dependency link from a global variable in the job flow and select Edit.
The Edit Global Variable Dependency dialog opens.
3. Complete the following fields:

Value

Specifies the value to which the global variable is compared.

[Operator]

Specifies the operator to use when comparing the specified value with the current value of the global variable, as follows:

=

Equal. If the global variable is equal to the value specified, processing continues with the next step in the flow.

!=

Not equal. If the global variable is not equal to the value specified, processing continues with the next step in the flow.

<

Less than. If the global variable is less than the value specified, processing continues with the next step in the flow.

>

Greater than. If the global variable is greater than the value specified, processing continues with the next step in the flow.

4. Click OK.

The global variable dependency value is set.

Commit Changes in Application Editor

You must commit job property and job flow changes to the appropriate CA Workload Automation AE server to make the changes permanent on that server.

Note: You must save job property changes before you perform a commit; however, it is not necessary to save the Application Editor workspace before you perform a commit.

To commit changes in Application Editor

1. Make the required changes in Application Editor.
2. Select Commit to scheduler from the drop-down list in the Flow section, and click Go.

The Commit Jobs dialog opens.

3. Select the server you want to commit the changes to, select the appropriate options, and click OK.

Note: You must select Process Jobs in Delete List to delete the jobs and box jobs in the Delete List on the specified server.

The Job Commit Progress dialog appears and the changes are committed to the scheduler.

4. Click Close when the commit process completes.

The Job Commit Progress dialog closes.

Create a Job in Quick Edit

You can create jobs to manage and schedule workload in your enterprise.

To create a job in Quick Edit

1. Open Quick Edit.
2. Click Create.

The Create Object dialog opens.

3. Select the appropriate server from the Server drop-down list. The server you select is where you want to store the job.
4. Click the link for the type of job you want to create.

Note: You can select a specific job type or, if available, use a job template containing pre-defined properties to create a new job.

The Properties section for the job appears.

5. Define your job by doing the following:
 - a. Enter a name for the new job in the Name field.
 - b. Complete the fields in each category as appropriate.
6. Click Commit.

The job is saved in the scheduler database.

Note: The Job Flow section is not available. The job flow displays only when viewing or modifying a job, not when creating one.

More information:

[Quick Edit and Application Editor](#) (see page 11)

Create a Global Variable in Quick Edit

You can use Quick Edit to define and store global variables in the CA Workload Automation AE database. You can then use the global variable within a job definition to set up a dependency condition.

To create a global variable

1. Open Quick Edit.
2. Click Create.
The Create Object dialog opens.
3. Select the appropriate server from the Server drop-down list. The server you select is where you want to store the global variable.
4. Click the Global Variable link from the Object Type list.
The Properties section for the global variable appears.
5. Specify the following properties:

Name

Specifies a name for the global variable.

Limits: Up to 64 characters

Value

(Optional) Specifies a value for the global variable.

Limits: Up to 100 characters

6. Click Commit.

The global variable is saved in the scheduler database.

More information:

[Global Variable Dependencies](#) (see page 44)

Commit Changes in Quick Edit

You must commit job property and job flow changes to the appropriate CA Workload Automation AE server to make the changes permanent on that server.

To commit changes in Quick Edit

1. Make the required changes to the job properties in Quick Edit.
2. Select the server you want to commit the job to.
3. Click Commit.

A confirmation message appears.

Chapter 4: Common Job Properties

This section contains the following topics:

- [Specifying the Job Owner](#) (see page 39)
- [Define Starting Conditions for a Job](#) (see page 40)
- [Define Resource Dependencies for a Job](#) (see page 50)
- [Defining Date and Time Dependencies](#) (see page 53)
- [Specifying Environment Variables](#) (see page 63)
- [Defining Job Termination Conditions](#) (see page 70)
- [Setting Up Notifications](#) (see page 75)
- [Specify the Users with Edit and Execute Permissions](#) (see page 82)

Specifying the Job Owner

By default, all jobs run under *authenticated_user@host*, where *authenticated_user* is the operating system user who invoked 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.

Important! 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.

For jobs that run on other software, such as PeopleSoft, the owner can be the user ID associated with and authenticated by the software. For example, PeopleSoft jobs run under PeopleSoft user IDs.

To specify the job owner, use the Owner property, located under the Primary category in the Properties section of a job definition.

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 property in job definitions. The job runs under the prod user on the unixagent computer.

To specify a job owner

1. Create a CMD job.
2. Enter the following Primary properties:
 - Name—jobA
 - Send to machine—UNIXAGENT
 - Owner—prod@unixagent
3. Enter the following Command property:
 - Command—/bin/touch /tmp/test_run.out
4. Save the job.

Define Starting Conditions for a Job

You can define dependencies to use as starting conditions for a job. CA Workload Automation AE uses Boolean logic to evaluate these dependencies. The job cannot run until all specified dependencies evaluate to true. You can specify one or more of the following dependencies:

- [Job status dependencies](#) (see page 41)
- [Cross-instance job dependencies](#) (see page 42)
- [Exit code dependencies](#) (see page 42)
- [Look-back dependencies](#) (see page 43)
- [Global variable dependencies](#) (see page 43)

You specify job dependencies using the Condition property, located under the Primary category in the Properties section of a job definition.

Note: When you specify a condition for an undefined job, the condition evaluates as false and any jobs dependent on the condition do not run.

More information:

- [Starting Condition Examples](#) (see page 45)
[Primary Properties](#) (see page 47)

Job Status Dependencies

You can define a starting condition for a job so that it starts when another job returns a specific status. For example, you might specify that JobB starts when JobA returns a SUCCESS status and JobC starts when JobB returns a SUCCESS status.

To define a starting condition based on a job's CA Workload Automation AE status, use the following format in the Condition property:

`status(job_name)`

status

Specifies one of the following:

Done

Indicates that the job you are defining may run when *the job_name*'s status is SUCCESS, FAILURE, or TERMINATED.

Failure

Indicates that the job you are defining may run when *job_name*'s status is FAILURE.

Notrunning

Indicates that the job you are defining may run when *job_name*'s status is anything except RUNNING. Set status to notrunning to keep the dependent job from running at the same time as *job_name*.

Success

Indicates that the job you are defining may run when *job_name*'s status is SUCCESS.

Note: You can use the Max exit code for success property, located under the Termination category, to control the value of the SUCCESS status for a specific job. When you define the max exit code for success, a job that exits with an exit code less than or equal to the specified value is treated as a success. FAILURE means the job exited with an exit code greater than the max exit code for success value.

Terminated

Indicates that the job you are defining may run when *job_name*'s status is TERMINATED. A status of TERMINATED means the job was killed.

Notes:

- You can use either uppercase or lowercase characters to define status condition identifiers.
- You can abbreviate the status condition identifiers with the first letter (d, f, n, s, and t). You can specify these abbreviations in uppercase or lowercase.

job_name

Identifies a job on which the job you are defining depends.

Cross-Instance Job Dependencies

To specify a cross-instance job dependency, enter the job name followed by a caret (^) and the name of another instance in the Condition property. For example, the following condition statement indicates that the dependent job can start when jobB on the CA Workload Automation AE instance PRD and jobA both return SUCCESS:

```
success(jobA) AND success(jobB^PRD)
```

Exit Code Dependencies

You can base job dependencies on exit codes that indicate completed tasks. For example, you can specify that the system should run JobB if JobA fails with an exit code of 4.

To define a starting condition based on a job's exit code, use the following format in the Condition property:

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

job_name

Identifies a job on which the job you are defining depends.

operator

Specifies one of the following comparison operators:

- = Equals
- != Does not equal
- < Less than
- > Greater than
- <= Less than or equal to
- >= Greater than or equal to

value

Defines the exit code value to compare against.

Look-Back Dependencies

The look-back feature is defined as the last run of the condition or predecessor job. The last run is equivalent to the ending time of the last successful run of the job. If the condition or predecessor job has run successfully after the job for which that condition or predecessor job is defined, then the dependency is satisfied and the job starts. Otherwise, the dependency is not satisfied and the job is not started.

To support the look-back feature, you must add an additional time operand to the Condition statement. You can apply the look-back feature to job status, cross-instance or external dependency job status, and exit code condition types only. You cannot apply it to the global variable condition type. To define a starting condition based on a look-back value, use one of the following formats in the Condition property:

```
status(job_name, hhhh:mm)
status(job_name^INS, hhhh:mm)
exitcode(job_name, hhhh:mm) operator value
exitcode(job_name^INS, hhhh:mm) operator value
```

Note: Alternatively, you can use a colon instead of a period between the specified hours and minutes, but it must be escaped (:\:).

hhhh

Indicates the hours taken for the last run of the condition or predecessor job.

Limits: One to four numeric characters from 0 - 9999, which makes it possible to look back approximately 416.58 days

Note: When specifying minutes also, the valid values range from 0 – 9998 instead.

mm

Indicates the minutes taken for the last run of the condition or predecessor job. Periods or escaped colons are used to separate minutes from hours.

Limits: One to two numeric characters from 0 to 59, representing minutes

Examples: 0, 00.15, 06.30, 23.30, 24, 98.30, 720, 9998.59, and 9999

Notes:

- When the elapsed time to be specified is less than one hour, you must specify the hourly value as "00". For example, "00.30" represents 30 minutes. If hhhh.mm is specified as "30", it implies an elapsed time of 30 hours. Specifying an elapsed time of ".30" is not valid.

- When you specify 0, CA Workload Automation AE examines the last end time of the job first. It then examines the last end time of the condition or predecessor job. If the condition or predecessor job has run since the last run of the job for which it is a dependency, that job is allowed to start. If the condition or predecessor job has not run since the last run of the job for which it is a dependency, that job is not allowed to start.

Global Variable Dependencies

You can base job dependencies on global variables set using the `sendeevent` command. When using global variables, the value of the variable must evaluate to true to satisfy the job dependency.

To define a starting condition based on a global variable, use the following format in the Condition property:

`VALUE(global_name) operator value`

global_name

Identifies a global variable that is already defined.

Limits: Up to 30 alphanumeric characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), pound (#), and hyphen (-)

operator

Specifies one of the following comparison operators:

= Equals

!= Does not equal

< Less than

> Greater than

<= Less than or equal to

>= Greater than or equal to

value

Defines the value to compare against.

Limits: Up to 100 alphanumeric characters; cannot contain quotation marks or spaces

Starting Condition Examples

The following examples provide sample starting conditions for jobs.

Example: Define Starting Conditions Based on Job Success

This example (specified in the job definition for JobA) runs JobA if JobB succeeds:
`success(JobB)`

Example: Define Starting Conditions Based on the Status of Multiple Jobs

This example (specified in the job definition for JobC) starts JobC only when both JobA and JobB complete successfully *or* when both JobD and JobE complete (regardless of whether they failed, succeeded, or terminated):

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

Example: Define Starting Conditions Based on Job Failure

This example (specified in the job definition for JobA) runs JobA if JobB fails:
`failure(JobB)`

Example: Define Starting Conditions Based on Job Success in a Given Duration

This example (specified in the job definition for JobA) runs JobA if JobB has completed successfully in the last 12 hours:

`success(JobB,12)`

Example: Define Starting Conditions Based on the Status of Multiple Jobs in a Given Duration

This example (specified in the job definition of JobA) runs JobA if JobB has failed in the last 30 minutes and JobC has completed successfully on external instance XYZ in the last 90 minutes:

`failure(JobA,00.30) AND success(JobC^XYZ,01.30)`

Example: Define Starting Conditions with the notrunning Status Condition

You may want to use the notrunning status condition to avoid a database dump (DB_DUMP) and a file backup (BACKUP) from running at the same time, which would cause frequent hard disk accesses. This example (specified in the job definition for JobA) runs JobA only when both of these resource-intensive jobs are **not** running:

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

Example: Define Starting Conditions Based on Multiple Condition Types

This example (specified in the job definition for JobA) runs JobA only if the job BACKUP completes with a SUCCESS status and the global variable TODAY has a value of Friday:

```
success(BACKUP) AND value(TODAY)=Friday
```

Example: Define Starting Conditions Based on a Cross-Instance Job Status

This example (specified in the job definition for JobA) runs JobA only when the job DB_BACKUP (which resides on another CA Workload Automation AE instance named PRD) succeeds:

```
success(DB_BACKUP^PRD)
```

Example: Define Starting Conditions Based on Job Exit Code

This example (specified in the job definition for JobB) runs JobB when JobA completes with an exit code of 4:

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

Example: Specify a Look-Back Dependency

The following Condition statements let the dependent job start if JobA is successful during the last hour and JobB failed during the last 2 hours and 15 minutes:

```
success(JobA,01.00) and failure(JobB,02.15)  
success(JobA,01\:00) and failure(JobB,02\:15)
```

Example: Define Starting Conditions Based on a Global Variable Value

This example (specified in the job definition for JobA) runs JobA only if the value of the global variable OK_TO_RUN is greater than 2:

```
VALUE(OK_TO_RUN)>2
```

Primary Properties

The Primary category includes the following properties:

Application

(Optional) Defines the name of the application with which to associate the job. You can only associate a job with one application, but an application may have many jobs associated with it.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), pound (#), and hyphen (-); do not include embedded spaces or tabs

JIL attribute: application

Box

(Optional) Identifies an existing Box job in which to put the job.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), and hyphen (-)

JIL attribute: box_name

Condition

(Optional) Defines any combination of dependencies based on job status, job exit codes, or global variables.

Limits: Up to 4096 characters

Notes:

- To configure complex conditions, combine a series of conditions with the AND or the 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. The parentheses force precedence, and the equation is evaluated from left to right. For example, the following condition statement lets the dependent job start when either JobA and JobB return SUCCESS or JobD and JobE return SUCCESS, FAILURE, or TERMINATED:

(success(JobA) AND success(JobB)) OR (done(JobD) AND done(Job E))

JIL attribute: condition

Description

(Optional) Defines an optional free-form text description of the job.

Limits: Up to 500 alphanumeric characters (including spaces)

JIL attribute: description

Group

(Optional) Defines the name of the group with which to associate the job. You can only associate a job with one group, but a group may have many jobs associated with it.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), pound (#), and hyphen (-); do not include embedded spaces or tabs

JIL attribute: group

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

JIL attribute: insert_job

Owner

(Optional) Defines the owner of the job. By default, the owner property is set to *authenticated_user@host*, where *authenticated_user* is the operating system user who invoked JIL to define the job. For jobs that run on other software, such as PeopleSoft, the owner can be the user ID associated with and authenticated by the software. For example, PeopleSoft jobs run under PeopleSoft user IDs. If your job requires a different user ID, you must override the owner value. Otherwise, the job does not run successfully.

Notes:

- 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.

Limits: Up to 145 characters; case-sensitive; cannot contain delimiters (such as spaces)

JIL attribute: owner

Resources

(Optional) Specifies virtual and real resource dependencies.

Notes:

- To specify a virtual resource dependency, the virtual resource must already be defined on the database. Otherwise, you receive an error.
- You cannot define a resource dependency on a box.

Limits: Up to 4096 characters

JIL attribute: resources

Send to machine

Specifies the name of the computer where the agent runs.

JIL attribute: machine

Define Resource Dependencies for a Job

In the CA Workload Automation AE environment, a resource is a system component that can be quantified. Resources govern job execution and play a vital role in scheduling. Before starting a job, CA Workload Automation AE compares the resources required with the resources available. For example, you may have specified resource requirements that must be met before a job can run. If the required resources are not available, CA Workload Automation AE defers the job run until resources are available. There are two types of resources: real and virtual.

Real resources are resources that are physically associated with the system, such as memory, disk space, and CPU time. For example, consider a job that can run only when the available memory is greater than a specified value. In that case, CA Workload Automation AE starts the job only when the required memory is available.

Virtual resources are only representative and are not tied to a physical system. The number of licenses available to all machines controlled by CA Workload Automation AE is an example of a virtual resource. As each job that requires a license is submitted to the system, the number of licenses in the pool is decremented. When the number of licenses reaches zero, no more jobs can be submitted until the license pool is replenished.

You can define a job to have dependencies on virtual and real 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.

To specify real and virtual resource dependencies, use the following format for the Resources property in the Primary section of a job definition:

```
(virtual_resource_name, quantity=amount | ALL [, free=Y|N|A]) AND  
(real_resource_type, VALUEOP=operator, VALUE=res_value) [AND ...]
```

virtual_resource_name

Specifies the name of a virtual resource.

Note: The virtual resource must already be defined in the database.

quantity=amount | ALL

Specifies the amount of virtual resource required. The following options are available:

- *amount*—Specifies the number of units that the job requires. The number must be a positive integer.
- *ALL*—Specifies that the job requires all the units of the resource.

free=Y | N | A

(Optional for renewable virtual resources only) Specifies whether the units of the virtual resource are freed from the job. The following options are available:

- Y—Frees the units only if the job completes successfully. This is the default.
- N—Does not free the units.

Note: To free the resources, you must issue the following command:

```
sendevent -E RELEASE_RESOURCE -J job_name
```

- A—Frees the units unconditionally.

real_resource_type

Specifies the type of real resources. The following options are available:

- SYSTEM_CPU_COUNT—Defines the total number of CPUs that the job requires.
- SYSTEM_OS_TYPE—Specifies the operating system name that the job requires.
- SYSTEM_CPU_SPEED—Defines the system clock speed (in MHz) that the job requires.
- SYSTEM_PHYSICAL_MEMORY—Defines the total amount of available physical memory (in MB) that the job requires.
- SOFTWARE_NAME—Specifies the software that the job requires.

Note: Real resources are predefined to CA Workload Automation AE and are managed by CA DCA Manager. You cannot define or update real resources using CA Workload Automation AE.

VALUEOP=operator

Specifies the operator for the real resource dependency. The following options are available:

- EQ—Equal to
- NEQ—Not equal to
- LT—Less than
- LTE—Less than or equal to
- GT—Greater than
- GTE—Greater than or equal to

VALUE=*res_value*

Specifies the value of the real resource specified in the *real_resource_type* keyword.

Note: You can define multiple real and virtual resource dependencies in a job definition. Separate each dependency with **AND**.

More information:

[Primary Properties](#) (see page 47)

Resource Dependency Examples

The following examples provide sample resource dependencies for jobs.

Example: Define a Virtual Resource Dependency

This example (specified in the Resources property of a job) defines a dependency on a virtual renewable resource. Before the job can start running, it needs all the units of the renew_res resource. After the job completes, the units of the renew_res resource are not freed from the job.

(renew_res, QUANTITY=ALL, free=N)

Example: Define Real and Virtual Resource Dependencies

This example (specified in the Resources property of a job) defines 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
- SYSTEM_PHYSICAL_MEMORY is greater than 2 GB

The free keyword is not specified for the virtual resource dependencies, so the resource units are freed only if the job completes successfully.

(D1, QUANTITY=1) AND (T1, QUANTITY=3) AND (R1, QUANTITY=4, FREE=Y) AND
(SYSTEM_OS_TYPE, VALUEOP=EQ, VALUE=AIX) AND (SYSTEM_PHYSICAL_MEMORY, VALUEOP=GT,
VALUE=2097152)

Defining Date and Time Dependencies

You can schedule 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 again at 5:00 p.m.

You can specify days of the week or actual dates, but you cannot specify both. 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 a specific time of day or hourly at a designated number of minutes past the hour.

You can set up one of the following types of time dependencies: absolute or 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 25th). The following Schedule properties define absolute time dependencies:

- Days: Run calendar
- Days: Run days
- Exclude calendar
- Run window
- Time: Times of day

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 Schedule, Termination, and Notification properties define relative time dependencies:

- (Schedule) Time: Minutes after each hour
- (Termination) Hours after successful completion to delete job
- (Termination) Minutes to wait before terminating
- (Notification) Alarms: Maximum run time (minutes)
- (Notification) Alarms: Minimum run time (minutes)

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 jobs might shift to adjust for the time change. Jobs that are not time-dependent run as appropriate. During the time change, absolute times behave differently than relative times.

More information:

[Time Zone](#) (see page 54)
[Custom Calendars](#) (see page 54)
[Date and Time Dependency Examples](#) (see page 58)
[Schedule Properties](#) (see page 60)

Time Zone

By default, jobs with time-based starting conditions that do not specify a time zone are scheduled to start based on the time zone under which the CA Workload Automation AE server is running.

The Time zone Primary property defines the time zone on which to base scheduling for the job you are defining. For example, if you define a start time of 01:00 for a job running on a machine in Denver, and set the Time zone property to San Francisco (which is in the Pacific time zone, one hour earlier than Denver), the job starts at 2:00 a.m. Denver time.

Custom Calendars

You can use Quick Edit to define any number of 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.

Must Start Times and Must Complete Times

You can define the time, or a list of times, that a job must start or complete by. If the job does not start or complete by a specified time, an alarm is issued. Defining must start times and must complete times is helpful when you want to be notified when a job has not started or completed. For example, a must start alarm can alert you to investigate whether the job's starting conditions are not satisfied or the job is stuck in the STARTING state. You use the Must start time and Must complete time Schedule properties to set these times.

Notes:

- To define absolute times, you must select the Date/Time conditions property and specify times in the Times of day property.
- You cannot define absolute times when the start_mins attribute is specified in the job definition. You will get an error if you define absolute times with start_mins.

Example: Specify Absolute Must Start Times

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

Date/time conditions—selected

Run days—all

Times of day—"10:00, 11:00, 12:00"

Must start time—"10:02, 11:02, 12:02"

Must complete time—"10:08, 11:08, 12:08"

Note: The number of must start times must match the number of start times. Otherwise, the job terminates. For example, the job terminates if it has one start time and two must start times, as follows:

Times of day—"10:00"

Must start time—"10:02, 12:02"

Example: Specify an Absolute Must Start Time on the Next Day

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

```
Date/time conditions=y  
Run days-all  
date_condition: 1  
start_times: "12:00"  
Must start time="34:10"  
Must complete time="34:12"
```

The must start time is calculated as follows:

```
start time in hh:mm format + time difference between the start time and the must start time  
= 0:00 + (time difference between 12:00 a.m. and 10:10 a.m. the next day)  
= 0:00 + 34:10  
= 34:10
```

Example: Specify an Absolute Must Start Time That Crosses Two Days

This example defines a job to run at 16:00 (4:00 p.m.). Suppose that you want the job to start by 6:10 a.m. two days later. You must specify 54:10 in the `must_start_times` attribute.

```
insert_job: job3  
job_type: CMD  
machine: localhost  
command: echo "hello"  
date_conditions: y  
days_of_week: all  
date_condition: 1  
start_times: "16:00"  
must_start_times: "54:10"  
must_complete_times: "54:12"
```

The must start time is calculated as follows:

```
start time in hh:mm format + time difference between the start time and the must start time  
= 16:00 + (time difference between 16:00 and 6:10 a.m. two days later)  
= 16:00 + 38:10  
= 54:10
```

Example: Specify Relative Must Start Times

This example defines a job to run at 10:00 a.m., 11:00 a.m., and 12:00 p.m. The job must start within 3 minutes after each start time (10:03 a.m., 11:03 a.m., and 12:03 p.m.). Otherwise, an alarm is issued for each missed start 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 Times With start_mins

This example defines a job to run at every 10 minute interval in each other (for example, 2:00 p.m., 2:10 p.m., 2:20 p.m., and so on). The job must start within 2 minutes after the specified start times. Otherwise, an alarm is issued for each missed start time. For instance, the job must start by 2:12 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
```

Example: Specify a Relative Must Start Time on the Next Day

This example specifies a relative must start time of 12:01 a.m. the next day. The must start time is calculated by adding three minutes to the start time. The calculated time crosses into the next day.

```
insert_job: job2
job_type: CMD
machine: localhost
command: echo "Hello"
date_conditions: y
days_of_week: all
date_conditions: 1
start_times: "23:58"
must_start_times: +3
```

Example: Specify an Unmatched Number of Start and Complete Times

This example defines a job to run twice. Events for the must start time and must complete time are inserted to database. The job's first run occurs at 12:00 p.m. If the job runs for three minutes, the MUST_START_ALARM is not generated because the job started by the specified must start time (12:01). However, the MUST_COMPLETE_ALARM is issued because the job ran past the specified must complete time (12:02).

After the job's first run, the job terminates with an error because the job is scheduled for a second run, but a second set of must start time and must complete time is not specified. The number of must start times and must complete times must match the number of start times.

```
insert_job: job2
job_type: CMD
machine: localhost
command: echo "Hello"
date_conditions: y
days_of_week: all
date_conditions: 1
start_times: "12:00, 13:00"
must_start_time: "12:01"
must_complete_time: "12:02"
```

Date and Time Dependency Examples

The following examples set time dependencies using Schedule properties within a job definition.

Example: Set Time Dependencies for a Job

This example sets the existing job test_run to run automatically on certain days at 10:00 a.m. and 2:00 p.m. on Mondays, Wednesdays, and Fridays.

To set time dependencies for a job

1. Open the text_run job.
2. Specify the following Schedule properties:
 - Date/Time conditions—selected
 - Days—run days
 - Run days—Mon, Wed, Fri
 - Times of day—"10:00, 14:00"
3. Commit the job.

Example: Base Time Settings on a Specific Time Zone

You can use the Time zone field to base the time settings for a job on a time zone other than the default. 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:

Time zone—"IST-5:30"

Example: Run a Job Every Day

To run the job every day, instead of only on specific days, specify ALL for the Run days property as in the following example, instead of selecting the individual day values:

days_of_week: all

Example: Schedule a Job to Run on Specific Dates

To schedule a job for specific dates, instead of days of the week, specify a run calendar. Define the calendar and then use the Run calendar field to specify the calendar name in the job definition.

For example, specify the following Schedule properties to schedule a job to run on the days defined in the weekday_cal calendar:

- Days—Run calendar
- Run calendar—weekday_cal

Example: Exclude a Job from Running on Specific Dates

You can specify a custom calendar that defines days on which a job should not run, such as business holidays. Define the calendar and use the Exclude calendar field to specify the calendar name in the job definition. For example, specify the following Schedule property to exclude days defined in the holiday_cal calendar:

Exclude calendar—holiday_cal

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

To run a job at specific times every hour instead of at specific times of the day, use the Time and Minutes after each hour fields. For example, enter the following Schedule properties to run a job at 15 minutes after and 15 minutes before each hour:

- Time—Minutes after each hour
- Minutes after each hour—15, 45

Schedule Properties

The Schedule category includes the following properties:

Auto hold job when box starts

(Optional) Indicates whether to place an automatic hold on the job when its containing box starts. Selecting the check box places an automatic hold on the job. Clearing the check box removes the auto hold from the job.

JIL attribute: auto_hold

Average run time

(Optional) Specifies the average run time (in minutes) to use for a job that is newly submitted to the database.

JIL attribute: avg_runtime

Date/Time conditions

(Optional) Indicates whether to use date or time conditions specified to determine when to run the job.

JIL attribute: date_conditions

Days: Run calendar

(Optional) Specifies the calendar to define the schedule for running the job.

Limits: Up to 30 characters

JIL attribute: run_calendar

Days: Run days

(Optional) Specifies the days to run the job.

JIL attribute: days_of_week

Exclude calendar

(Optional) Specifies whether to use the calendar selected from the associated list as an *exclude* calendar at run time. An exclude calendar specifies days on which the job should not run.

Limits: Up to 30 characters

JIL attribute: exclude_calendar

Job load

(Optional) Defines the relative load of the job and can be any user-defined value. The value should have some relationship to the value defined by the max_load attribute.

Default: No load is assigned.

Note: The max_load attribute is the value of a machine and not of a job.

JIL attribute: job_load

Must complete times

(Optional) Defines the amount of time that a job is allowed to run or a specific time by which a job must run before completing. You can specify a relative value to the start time (Minutes after each hour) or an absolute value. If the job has not completed before the time expires, an alarm is issued. Relative times must be preceded by a plus sign (+). Absolute times must be specified in hh:mm format, where hh denotes hours (in 24-hour format) and mm denotes minutes. To specify multiple absolute times, separate each value by a comma.

Limits: Up to 255 characters

Examples: +5 (relative time), 11:30, 12:30, 15:30 (absolute times)

JIL attribute: must_complete_times

Must start times

(Optional) Defines the amount of time that a job is allowed to wait or a specific time until which a job must wait before starting. You can specify a relative value to the start time (Minutes after each hour) or an absolute value. If the job has not started before the time expires, an alarm is issued. Relative times must be preceded by a plus sign (+). Absolute times must be specified in hh:mm format, where hh denotes hours (in 24-hour format) and mm denotes minutes. To specify multiple absolute times, separate each value by a comma.

Limits: Up to 255 characters

Examples: +5 (relative time), 11:30, 12:30, 15:30 (absolute times)

JIL attribute: must_start_times

Priority

(Optional) Defines the queue priority of the job. The lower the value, the higher the priority; therefore, 1 is the highest possible queued priority. 0 signifies to run the job immediately, regardless of the current machine load

Limits: Any integer 0 or greater

Default: 0

JIL attribute: priority

Run window

(Optional) Specifies the interval during which a job may start in the format *hh:mm-hh:mm*, where hh denotes hours (in 24-hour format) and mm denotes minutes. The specified interval can overlap midnight, but it cannot encompass more than 24 hours.

Limits: Up to 20 characters

JIL attribute: run_window

Time: Times of day

(Optional) Specifies the times to start the job in *hh:mm* format, where hh denotes hours (in 24-hour format) and mm denotes minutes. To specify multiple times, separate each value by a comma.

Limits: Up to 255 characters

Example: 12:00, 00:00 runs the job at noon and midnight

JIL attribute: start_times

Time: Minutes after each hour

(Optional) Specifies the number of minutes after each hour, in *mm* format, to start the job. To specify multiple values, separate each value by a comma.

Limits: Up to 255 characters

Example: 00, 15, 30, 45 runs the job every 15 minutes

JIL attribute: start_mins

Time zone

(Optional) Specifies the time zone for the job. The job's time settings are based on this time zone, regardless of the server location. Specify a string that corresponds to an entry in the ujo_timezones table. On UNIX, you can specify any time zone recognized by the operating system.

UNIX Limits: Up to 50 characters; valid characters are a-z, A-Z, decimal digits, slash (/), hyphen (-), and underscore (_); not case-sensitive

Windows Limits: Do not include spaces or periods; not case-sensitive

Examples: IST-5:30, MountainTime (instead of Mountain Time), SolomonIs (instead of Solomon Is)

JIL attribute: timezone

Times to restart job after failure

(Optional) Specifies how many attempts should be made to restart the job after failure.

Limits: 0-200

JIL attribute: n_retrys

Specifying Environment Variables

Environment variables define the local environment where a script, command, or batch file runs. A profile defines the non-system environment variables that a job uses. You specify environment variables and profiles using the Environment variables and Profile properties, located under the Environment category in the Properties section of a job definition.

More information:

[Environment Variables Examples](#) (see page 63)

[Job Profile Examples](#) (see page 66)

[Environment Properties](#) (see page 67)

Environment Variables Examples

The following examples provide samples of using environment variables on UNIX and Windows.

Example: Pass UNIX Environment Variables to a Script

This example passes two environment variables to a script and a third environment variable that defines the Present Working Directory (PWD). In this example, the pay script can reference these variables:

| Environment Variable | Value Passed |
|----------------------|-----------------------|
| NAME | user 1 |
| JOB | PAYROLL |
| PWD | /usr/scripts/dailyrun |

To pass UNIX environment variables to a script

1. Create a Command job.
2. Enter the following properties:
 - Name—UNIX_JOB
 - Send to machine—UNIX_NY
 - Command—/home/scripts/pay
 - Environment variables—NAME=user
1,JOB=PAYROLL,PWD=/usr/scripts/dailyrun
3. Commit the job.

Example: Set the User's Profile on UNIX

This example sets the user's profile my_profile in their home directory /usr/home:

To set the user's profile on UNIX

1. Create a Command job.
2. Enter the following Environment property:
 - Job environment—/usr/home/my_profile
3. Commit the job.

Example: Pass Windows Environment Variables to a Batch File

This example passes three environment variables to a Windows batch file and a fourth environment variable that defines the home directory. In this example, the command command1 can reference these variables:

| Environment Variable | Value Passed |
|----------------------|--------------|
| A | B |
| C | 'X Y' |
| E | pay |
| HOME | c:\export\u1 |

To pass Windows environment variables to a batch file

1. Create a Command job.
2. Enter the following properties:
 - Name—NT_JOB
 - Send to machine—NT_NY
 - Command—c:\payroll\command1
 - Environment variables—A=B,C=X Y,E=pay,HOME=c:\export\u1
3. Commit the job.

Example: Set the Windows Environment From the Profile

This example sets the environment from the profile eng (which was previously defined in the Job Profiles Manager):

To set the Windows environment from the profile

1. Create a Command job.
2. Enter the following Environment property:
 - Job environment—eng
3. Commit the job.

Example: Set the Environment From the Profile Which is Different From the Machine on Which the Job Runs

This example sets the environment from the profile eng that was defined on the machine dev, which is different from the machine on which the job runs:

To set the Windows environment from the profile

1. Create a Command job.
2. Enter the following Environment property:
 - Job environment—\\dev\\eng
3. Commit the job.

More information:

[UNIX Environment Variables](#) (see page 105)
[Environment Properties](#) (see page 67)

Job Profile Examples

The following examples provide samples of using job profiles on UNIX and Windows.

Example: Set the Environment From the Profile

This example sets the environment from the profile eng (which was previously defined in the Job Profiles Manager):

```
profile: eng
```

Example: Set the Environment From the Profile Which is Different From the Machine on Which the Job Runs

This example sets the environment from the profile eng that was defined on the machine dev, which is different from the machine on which the job runs:

```
profile: \\dev\eng
```

Example: Set the User's Profile

This example sets the user's profile my_profile in their home directory /usr/home:

```
profile: /usr/home/my_profile
```

More information:

[Environment Properties](#) (see page 67)

[UNIX Environment Variables](#) (see page 105)

Environment Properties

Note: This section will not display for certain job types that do not have Environment properties.

The Environment category includes the following properties:

Environment variables

(Optional) Specifies environment variables to define the local environment where the script, command, or batch file runs. You can modify existing environment variables or create your own.

This field uses the following format:

parm_name=parm_value[, parm_name=parm_value...]

parm_name

Defines the name of a new environment variable or specifies the name of an existing environment variable.

Note: Enclose names that contain spaces in single quotation marks.

Example: 'exceptions 1'

parm_value

Specifies the value of the environment variable.

Note: Enclose names that contain spaces in single quotation marks.

Example: 'cmd1 cmd2 cmd3'

JIL attribute: envvars

File system check

(Optional) Defines the minimum amount of file space that must be available on designated drives for the job to start.

Windows:

- Specify one or more drives (by drive letter) and their corresponding sizes in kilobytes (KB). Only drives are checked; directories, if specified, are ignored.
- When specifying drive letters, you must escape the colon with double quotation marks or backslashes.

Example: "C: 100 D: 120"

UNIX:

- Specify one or more file systems (with full path names or directory names) and their corresponding sizes in kilobytes (KB).

Example: /roots 100 /auxfs1 120

Limits: Up to 255 characters for the entire value

Note: If you specify multiple drive or file system/size pairs, separate each with a single space.

JIL attribute: chk_files

Profile

(Optional) Specifies a job profile that defines environment variables to set before executing the specified command. Enter the name of a profile located on the job's client or a path name using the following format that includes the machine on which the profile resides: *machine_name*\profile_name

Limits: Up to 255 characters

Default: If you omit this field, the scheduling manager uses the "default" profile on Windows and the /etc/auto.profile on UNIX.

Note: This attribute is only available for Command, User Defined, and File Watcher job types.

JIL attribute: profile

Ulimit

(Optional) Specifies the UNIX shell's resource usage limit using the following format:

```
resource_type="soft_value,hard_value"[,  
resource_type="soft_value,hard_value"]...
```

resource_type

Specifies the resource type. Options are the following:

- c—Specifies the core file size in blocks.
- d—Specifies the data segment size in kilobytes (KB).
- f—Specifies the maximum file size in blocks.
- m—Specifies the process virtual size in kilobytes (KB).
- n—Specifies the number of files.
- s—Specifies the stack size in kilobytes (KB).
- t—Specifies the CPU time in seconds.

soft_value

Defines the usage (soft) limit for the specified resource type. The soft limit can be increased up to the specified hard limit and can be changed without root authority.

Limits: Any numeric digits or you can specify **unlimited**; the value must be less than or equal to the hard limit

hard_value

Defines the maximum usage (hard) limit for the specified resource type. The hard limit can only be increased by the root user. Any user ID can decrease a hard limit.

Limits: Any numeric digits or you can specify **unlimited**; the value must be greater than or equal to the soft limit

Note: If the job runs under a non-root user ID and the value specified in the job definition is greater than the current hard limit on the UNIX system, the hard limit will not be increased.

Example: c="100,200", s="250,300"

JIL attribute: ulimit

Defining Job Termination Conditions

You can specify job termination conditions using the Termination section of a job definition. For example, you can specify exit codes that CA Workload Automation AE interprets as job success or the maximum number of minutes a job can run before it is terminated.

You can specify one or more of the following job termination conditions:

- [Exit codes to indicate job failure or success](#) (see page 70)
- [Maximum runtime](#) (see page 72)
- [Automatic deletion when a job completes](#) (see page 72)
- [Termination of a failed box job or containing box](#) (see page 73)

Exit Codes to Indicate Job Failure or Success

By default, the scheduling manager interprets an exit code of zero (0) 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. You can also specify a maximum exit code the job can finish with to still be considered successful.

To define exit codes to indicate job failure or success using the following Termination properties within a job definition:

- Failure exit codes
- Success exit codes
- Max exit code for success

Note: These properties apply to Command, i5/OS, Micro Focus, z/OS Regular, and z/OS Manual jobs.

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 range of 40-50.

To define a range of exit codes to indicate job failure

1. Create a CMD job.
2. Enter the following Primary properties:
 - Name—cmdjob
 - Send to machine—AGENT
3. Enter the following Command property:
 - Command—"c:\temp\procjob.exe"
4. Enter the following Termination property:
 - Failure exit codes—40-50
5. Save the job.

Example: Define an 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 exit codes property defines exit code 100 as success, indicating successful completion of the script.

To define an exit code to indicate job success

1. Create a CMD job.
2. Enter the following Primary properties:
 - Name—test_blob
 - Send to machine—AGENT
3. Enter the following Command property:
 - Command—/home/esp/payroll.sh
4. Enter the following Termination property:
 - Success exit codes—0,100
5. Save the job.

More information:

[Termination Properties](#) (see page 73)

Maximum Runtime

The Minutes to wait before terminating Termination property specifies the maximum run time (in minutes) that the job you are defining should require to finish normally. If the job runs longer than the specified time, CA Workload Automation AE terminates it.

Example: Terminate the Job if it Runs Beyond the Maximum Run Time

This example specifies that the product should terminate the job if it runs for more than two hours:

Minutes to wait before terminating—120

Automatic Deletion When a Job Completes

The Hours after successful completion to delete job property specifies whether to automatically delete the job after completion. If the job is a box, the product deletes the box and all the jobs it contains.

This property is useful for scheduling and running a one-time batch job. You can specify the interval after the job's completion at which to delete the job.

Example: Delete a Job on Successful Completion

This example specifies that the product should automatically delete the job five hours after successful completion:

Hours after successful completion to delete job—5

Termination of a Failed Box Job or Containing Box

The Terminate containing box on job failure property, in the Termination section of a job definition, specifies whether to terminate the box job containing the job you are defining when the job completes with a FAILURE or TERMINATED status. This property only applies to jobs that are in a box.

The Terminate on failure of containing box property specifies whether to use a KILLJOB event to terminate the job if its containing box job completes with a FAILURE or TERMINATED status.

Use these two Termination properties to control how nested jobs react when a job fails.

Termination Properties

The Termination category includes the following properties:

Failure exit codes

(Optional) Defines which exit codes indicate job failure. Specify a single exit code (for example, 4), a range of exit codes (for example, 0-9999), a list of multiple exit codes separated by commas, or a list of ranges separated by commas.

Limits: Up to 256 characters

Default: Any exit code other than 0 (zero) indicates job failure.

Examples: -20--10,1,20-30,150

Note: If you specify multiple exit codes, enter the most specific codes first followed by the ranges.

JIL attribute: fail_codes

Hours after successful completion to delete job

(Optional) Specifies the maximum time (in hours) after job completion that the job should exist before being automatically deleted.

Limits: -1-596523

Default: -1

JIL attribute: auto_delete

Max exit code for success

(Optional) Defines the maximum exit code that the job can exit with and be considered a success.

Limits: 0-2,147,483,647

Default: 0

Note: CA Workload Automation AE reserves exit codes 120 and greater for internal use.

JIL attribute: max_exit_success

Minutes to wait before terminating

(Optional) Specifies the maximum run time (in minutes) for the job. If the job runs longer than the specified time, it is automatically terminated.

JIL attribute: term_run_time

Success exit codes

(Optional) Defines which exit codes indicate job success. Specify a single exit code (for example, 4), a range of exit codes (for example, 0-9999), or a list of multiple exit codes or ranges separated by commas.

Example: 1,20-30,150

Note: If you specify multiple exit codes, enter the most specific codes first followed by the ranges.

JIL attribute: success_codes

Terminate containing box on job failure

(Optional) Indicates whether to terminate the box job containing the job if the job fails or terminates.

JIL attribute: box_terminator

Terminate on failure of containing box

(Optional) Indicates whether to terminate the job if the box job it is in fails or terminates.

JIL attribute: job_terminator

Setting Up Notifications

Notifications let you automatically monitor a job's progress and notify users or take action when the job reaches a certain state or condition. You can specify properties within a job definition to generate an alarm, send an email notification, or create a service desk request. Notifications let you automate your workload processing.

You can specify one or more of the following notification actions:

- [Send an alarm](#) (see page 76)
- [Send a notification when a job completes](#) (see page 77)
- [Create a service desk request when a job fails](#) (see page 78)

Specify to Send an Alarm Under Certain Conditions

You can specify properties within a job definition to generate an alarm that alerts an operator to take corrective action. You can set up an alarm for when a job completes with a FAILURE or TERMINATED status, or when a job completes in less than or greater than a specified time.

To specify to send an alarm under certain conditions

1. Create a job in either Quick Edit or Application Editor.
The Properties section for the job appears.
2. Expand the Notification section.
3. Complete one or more of the following optional properties under the Alarms section:

Minimum run time (minutes)

Specifies the minimum run time for the job in minutes. When the job ends before the specified interval elapses, CA WCC generates an alarm.

Maximum run time (minutes)

Specifies the maximum run time for the job in minutes. When the job does not finish before the specified interval elapses, CA WCC generates an alarm.

Send alarm on failure

Specifies whether to generate an alarm if the job fails for any reason other than the minimum or maximum run time elapses.

4. Commit the job.

When the job completes under one of the defined conditions, CA Workload Automation AE generates an alarm.

More information:

[Notification Properties](#) (see page 79)

Specify to Send a Notification When a Job Completes

You can specify properties within a job definition to send a notification to someone when the job completes. You can specify whether CA Workload Automation AE sends a notification if the job completes with a FAILURE status or when the job completes with any status.

To specify to send a notification when a job completes

1. Create a job in either Quick Edit or Application Editor.
The Properties section for the job appears.
2. Expand the Notification section.
3. Complete the following Notification services properties:

ID

Specifies the notification ID or email address of the user who should receive the notification.

Message

Defines a message to include in the notification.

Notification

Specifies the conditions under which a notification is sent when a job completes.

4. Commit the job.

When the job completes with the status specified, CA Workload Automation AE sends a notification to the ID specified.

More information:

[Notification Properties](#) (see page 79)

Specify to Create a Service Desk Request When a Job Fails

You can specify properties within a job definition to create a service desk request when the job completes with a FAILURE status.

To specify to create a CA service desk request when a job fails

1. Create a job in either Quick Edit or Application Editor.
The Properties section for the job appears.
2. Expand the Notification section.
3. Select the Service desk request check box.
4. Complete the following optional Service desk request properties:

Description

Specifies a description to include in the CA Service Desk request generated when the job fails.

Priority

Specifies the priority level of the request.

Impact

Specifies the impact level of the request

Severity

Specifies the severity level of the request.

Note: Alternatively you can use the Additional information property to set attribute values. For more information, see the CA Service Desk documentation.

5. Commit the job.

When the job fails, CA Workload Automation AE sends a service desk request.

More information:

[Notification Properties](#) (see page 79)

Notification Properties

The Notification category includes the following properties:

Alarms: Maximum run time (minutes)

(Optional) Specifies the maximum run time for the job in minutes. When the job does not finish before the specified interval elapses, CA WCC generates an alarm.

Limits: 0-65535

JIL attribute: max_run_alarm

Alarms: Minimum run time (minutes)

(Optional) Specifies the minimum run time for the job in minutes. When the job ends before the specified interval elapses, CA WCC generates an alarm.

Limits: 0-65535

JIL attribute: min_run_alarm

Alarms: Send alarm on failure

(Optional) Specifies whether to generate an alarm if the job fails for any reason other than the minimum or maximum run time elapses.

JIL attribute: alarm_if_fail

Notification services: ID

(Optional) Specifies the notification ID or email address of the user who should receive the notification.

Limits: Up to 255 characters

JIL attribute: notification_id

Notification services: Message

(Optional) Defines a message to include in the notification.

Limits: Up to 255 characters

JIL attribute: notification_msg

Notification services: Notification

(Optional) Specifies the conditions under which a notification is sent when a job completes, as follows:

- Do not notify—Does not send a notification.
- Notify only if job fails—Sends a notification only if the job completes with a FAILURE status.
- Notify unconditionally—Sends a notification when the job completes with any status.

Default: Do not notify (on CA Workload Automation AE)

JIL attribute: send_notification

Service desk request: Additional information

(Optional) Specifies additional information to include in the service desk request generated when the job fails.

Limits: Up to 255 characters

JIL attribute: svcdesk_attr

Service desk request: Description

(Optional) Specifies a description to include in the service desk request generated when the job fails.

Limits: Up to 255 characters

JIL attribute: svcdesk_desc

Service desk request: Impact

(Optional) Specifies the impact level of the request, as follows:

- 0 - None
- 1 - High
- 2 - Medium-high
- 3 - Medium
- 4 - Medium-low
- 5 - Low

JIL attribute: svcdesk_imp

Service desk request: Priority

(Optional) Specifies the priority level of the request, as follows:

- 0 - None
- 1 - High
- 2 - Medium-high
- 3 - Medium
- 4 - Medium-low
- 5 - Low

JIL attribute: svcdesk_pri

Service desk request: Service desk request

(Optional) Indicates whether to open a service desk request when the job fails.

JIL attribute: service_desk

Service desk request: Severity

(Optional) Specifies the severity level of the request, as follows:

- 1 - Low
- 2 - Medium-low
- 3 - Medium
- 4 - Medium-low
- 5 - High

JIL attribute: svcdesk_sev

Specify the Users with Edit and Execute Permissions

Native security permissions specify which users have edit and execute permissions for the job you are defining.

Note: The job's owner always has full edit and execute permissions.

To specify the users with edit and execute permissions

1. Create a job in either Quick Edit or Application Editor.
The Properties section for the job appears.
2. Select one or more of the Native security permissions properties:
 - Group (execute)
 - Group (edit)
 - World (execute)
 - World (edit)
 - All hosts (execute)
 - All hosts (edit)
3. Save the job.

Example: Set Permissions to Execute a Job on Windows

This example sets permissions so that any authorized user can execute the job regardless of the machine they are logged on to. A Command job running on Windows is defined with the following native security permission selected:

All hosts (execute)

Example: Set Permissions to Execute and Edit a Job on UNIX

This example sets permissions so that any user can execute the job, but only members of your group can edit it. A Command job running on UNIX is defined with the following native security permissions selected:

- Group (edit)
- World (execute)

More information:

[Native Security Permissions Properties](#) (see page 83)

Native Security Permissions Properties

The Native Security Permissions category includes the following properties:

Permissions

Indicates that any authorized user may edit the job, regardless of the machine they are on. Otherwise, the user must be logged on to the machine specified in the owner field (for example, user@host_or_domain). Indicates the permission levels to associate with the job, as follows:

Group (execute)

Assigns group execute permissions to the job.

JIL attribute: permission: gx

Group (edit)

Assigns group edit permissions to the job.

JIL attribute: permission: ge

World (execute)

Assigns world execute permissions to the job.

JIL attribute: permission: wx

World (edit)

Assigns world edit permissions to the job.

JIL attribute: permission: we

All hosts (execute)

Indicates that any authorized user may execute the job, regardless of the machine they are on. Otherwise, the user must be logged on to the machine specified in the owner field (for example, user@host_or_domain).

JIL attribute: permission: mx

All hosts (edit)

Indicates that any authorized user may edit the job, regardless of the machine they are on. Otherwise, the user must be logged on to the machine specified in the owner field (for example, user@host_or_domain).

JIL attribute: permission: me

Chapter 5: Box Jobs

This section contains the following topics:

- [Box Jobs \(see page 85\)](#)
- [Define a Box Job \(see page 86\)](#)
- [Grouping Jobs and Boxes \(see page 88\)](#)
- [Add a Job to a Box Job \(see page 89\)](#)

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 let 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.

Define a Box Job

You can create a Box job to contain multiple jobs that are related by starting conditions or other criteria. 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.

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

To define a Box job

1. Create a Box (BOX) job in either Quick Edit or Application Editor.
The Properties section for the Box job appears.
2. Specify the following required property:

Name

Defines the name of the job that you want to schedule.

3. (Optional) Specify optional [Box properties](#) (see page 87).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Box job is defined.

More information:

[Add a Job to a Job Flow Using the Palette](#) (see page 26)
[Box Job Properties](#) (see page 87)

Box Job Properties

The Box category includes the following properties:

Failure

(Optional) Specifies the job failure conditions using a logical expression. The Box job is evaluated against these conditions. If no job failure condition is specified, the Box job fails if at least one job in the Box fails.

Limits: Up to 4096 characters

Notes:

- 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. The parentheses force precedence, and the equation is evaluated from left to right. For example, the following expression indicates that the Box job is considered to have failed when either JobA and JobB return FAILURE or JobD returns FAILURE (for example, if you set up JobD to attempt to recover a failed job):

`(f(JobA) AND f(JobB)) OR f(JobD)`

JIL attribute: box_failure

Success

(Optional) Specifies the job success conditions using a logical expression. The Box job is evaluated against these conditions. If no job success condition is specified, the Box job succeeds if all the jobs in the box complete successfully.

Limits: Up to 4096 characters

Notes:

- 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. The parentheses force precedence, and the equation is evaluated from left to right. For example, the following expression indicates that the Box job is considered successful when either JobA and JobB return SUCCESS or JobD and JobE return SUCCESS, FAILURE, or TERMINATED:

`(s(JobA) & s(JobB)) | (d(JobD) & d(Job E))`

JIL attribute: box_success

Grouping Jobs and Boxes

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 properties so you can logically group sets of jobs and boxes with unrelated starting conditions or dependencies. By specifying both Group and Application properties 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

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: c
...
group: HumanResources
application: EmployeePay
insert_job: ACCT_salaryreport
job_type: c
...
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
```

Add a Job to a Box Job

You can use a Box job to group jobs with like starting conditions such as date/time conditions or job dependency conditions. For example, if you have a number of jobs that run daily at 1:00 a.m., you can add all these jobs to a Box job and assign a daily start condition to the box.

To add a job to a Box job

1. Open the job you want to add using Quick Edit or Application Editor.

The Properties page for the job appears.

2. Specify the following Primary property to identify the Box job:

Box

Identifies an existing Box job in which to put the job.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), and hyphen (-)

3. Commit the job.

The job is added to the Box job.

Chapter 6: Command Jobs

This section contains the following topics:

- [Command Jobs \(see page 91\)](#)
- [Define a Command Job \(see page 92\)](#)
- [Verify File Space Before a Job Starts \(see page 102\)](#)
- [Pass Positional Arguments in a Command Job \(see page 103\)](#)
- [UNIX Environment Variables \(see page 105\)](#)
- [Define Alternative Error, Input, and Output Sources and Destinations \(see page 106\)](#)

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.

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.

Resource 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.

Define a Command Job

You can define a Command job to run workload on UNIX and Windows 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. Create a Command (CMD) job in either Quick Edit or Application Editor.
The Properties section for the Command job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer 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.

Limits: Up to 1000 characters; case-sensitive

Note (UNIX only): When you define a job that runs a script that calls a second script, the fully qualified path of the called script must be provided.

3. (Optional) Specify optional [Command properties](#) (see page 98).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Command job is defined.

More information:

[Add a Job to a Job Flow Using the Palette](#) (see page 26)

[Pass Positional Arguments in a Command Job](#) (see page 103)

[Define Alternative Error, Input, and Output Sources and Destinations](#) (see page 106)

Command Job Examples (UNIX)

The following examples describe sample Command jobs that run UNIX scripts and commands.

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 AGENT.

To run a command on UNIX

1. Create a Command job.
2. Enter the following properties:
 - Name—test_run
 - Send to machine—AGENT
 - Command—/bin/touch /tmp/test_run.out
3. Commit the job.

Example: Run a Script that is Located in a Path Set in the PATH Environment Variable

Suppose that the job PROCSCRIPT runs a script named procscript.sh. The job runs under jsmith, the user who has the authority to run the script. The path to procscript.sh is set in the PATH system environment variable for jsmith. This job runs on the default agent computer.

To run a script that is located in a path set in the PATH environment variable

1. Create a Command job.
2. Enter the following properties:
 - Name—PROSCRIPT
 - Owner—jsmith
 - Send to machine—AGENT
 - Command—procscript.sh
3. Commit the job.

Example: Run a Script that is Located in a Path Set in a User Environment Variable

Suppose that the job MYSCRIPTJOB runs a script named myscript.sh. The job runs under jsmith, the user who 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. This job runs on the default agent computer.

To run a script that is located in a path set in a user environment variable

1. Create a Command job.
2. Enter the following properties:
 - Name—MYSCRIPTJOB
 - Owner—jsmith
 - Send to machine—AGENT
 - Command—\$MY_PATH/myscript.sh
3. Commit the job.

Example: Run a Script that Uses the C Shell

In the following example, the C shell is used to run the sort script on the UNIX_LA agent computer.

To run a script that uses the C shell

1. Create a Command job.
2. Enter the following properties:
 - Name—MFGSORT
 - Send to machine—UNIX_LA
 - Command—/mfg/test/sort
 - Shell—/bin/csh
3. Commit the job.

Example: Send a User-Defined Exit Code

The following 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
```

In this job, the Success exit codes field defines exit code 100 as SUCCESS, indicating successful completion of the script.

To send a user-defined exit code

1. Create a Command job.
2. Enter the following properties:
 - Name—SUCCESSJOB
 - Owner—jsmith
 - Send to machine—SUN_NY
 - Command—/home/esp/payroll.sh
 - Success exit codes—0,100
3. Commit the job.

Command Job Examples (Windows)

The following examples describe sample Command jobs that run Windows command files.

Example: Run a Command File on Windows

This example runs the c:\bin\test.bat command on the Windows client computer named AGENT.

To run a command file on Windows

1. Create a Command job.
2. Enter the following properties:
 - Name—test_run
 - Send to machine—AGENT
 - Command—"c:\bin\test.bat"
3. Commit the job.

Example: Access a Windows Network Resource with Universal Naming Convention (UNC) and Share Names

Suppose that the path c:\WINNT\Profiles\Visitor\Desktop\ has the share name MyDesktop. The command notify.cmd is in that path on the CYBNT server, and is accessed by the UNC name and share name in the job JOBC, which runs on the agent WINAGENT.

Note: A share name is an alias for the path in which the resource exists. 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.

To access a Windows network resource with UNC and share names

1. Create a Command job.
2. Enter the following properties:
 - Name—JOBC
 - Send to machine—WINAGENT
 - Command—\\CYBNT\MyDesktop\notify.cmd
3. Commit the job.

Example: Use Administrative Authority to Access a Remote Windows Resource that is Not Shared

The agent service can log on to a remote Windows server as a user with administrative authority. Using this authority lets the agent access remote resources that are not marked as shared using the share names C\$ and ADMIN\$. Suppose that drive C is accessed by an administrator over the network through the WINAGENT agent. The agent runs under the System Account option, and it 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.

To use administrative authority to access a remote Windows resource that is not shared

1. Create a Command job.
2. Enter the following properties:
 - Name—SHAREFILE
 - Owner—admin1
 - Send to machine—WINAGENT
 - Command—\\CYBNT\C\$\working\test
3. Commit the job.

Example: Pass Arguments to a Visual Basic Script

Suppose that you want the job VBS to pass three arguments (one, two, and three) to the Visual Basic script located at D:\temp\vbs\params.vbs. To pass arguments to a Visual Basic script, specify the location of the Visual Basic script and its arguments in the Arguments to pass field. This job runs on the default agent computer.

To pass arguments to a Visual Basic Script

1. Create a Command job.
2. Enter the following properties:
 - Name—VBS
 - Send to machine—WINAGENT
 - Command—c:\winnt\system32\wscript.exe D:\temp\vbs\params.vbs one two three
3. Commit the job.

Command Properties

The Command Job category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Blob input

(Optional) Specifies the text to insert for the blob.

Limits: Up to 255 alphanumeric characters

JIL attribute: blob_input

Command

Specifies a command, executable, UNIX shell script, application, or batch file to run on the client using the following format:

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 positional 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.

Note (UNIX only): When you define a job that runs a script that calls a second script, the fully qualified path of the called script must be provided.

Limits: Up to 512 characters; case-sensitive

JIL attribute: command

Heartbeat interval

(Optional) Defines the frequency (in minutes) with which the job sends a heartbeat. The heartbeat is a signal that helps the scheduling manager monitor the progress of a job.

Default: 0 (The agent does not listen for heartbeats.)

JIL attribute: heartbeat_interval

Interactive

(Optional) Indicates whether the specified command needs to be user interactive.

JIL attribute: interactive

Shell

(Optional) Specifies the name of the shell used to execute the script or command file. You can specify any of the following UNIX shells:

- /bin/ksh (Korn shell)
- /bin/sh (Bourne shell)
- /bin/bash (Bourne again shell)
- /bin/csh (C shell)
- /usr/local/bin/perl (Perl shell)

JIL attribute: shell

Standard error file

(Optional) Defines the path and file name where you want to redirect all standard error output.

If you are running jobs across platforms, the scheduler of the issuing instance controls the default behavior. For UNIX, the default is to append this file. For Windows, the default is to overwrite this file. To overwrite the file, enter > as the first character in the value. To append the file, enter >> as the first characters in the value.

Limits: Up to 255 characters; it cannot contain spaces between the >> characters and the full path or file name

Notes:

- The job owner must have write permission to the specified file on the client computer.
- When specifying standard input, output, and error file locations, we recommend that you enter absolute paths to the files instead of relative paths or file names alone.
- You can use variables exported from the job profile or from global variables in the path name specification. Enclose variables referenced in the job profile in braces (for example, "\${PATH}") . Use the format \${global_name} for global variables.

UNIX:

- This value overrides the instance-wide setting for the AutoInstWideAppend parameter in the configuration file.
- This value overrides the machine-specific setting for the AutoMachWideAppend parameter in the /etc/auto.profile file.

Windows:

- If you do not specify a full path name, the path defaults to the job owner's home directory.
- This value overrides the instance-wide setting for the Append stdio/stderr field in the CA Workload Automation AE Administrator.
- When specifying drive letters in job definitions, you must escape the colon with double quotation marks or backslashes. For example, C:\\tmp or "C:\\tmp" is valid; C:\\tmp is not.
- If the Windows command specified in the job definition does not exist, the job does not run. The standard error file is not created and the job log indicates that the error file is not found.

Default: /dev/null (UNIX) or NULL (Windows)

JIL attribute: std_err_file

Standard input file

(Optional) Specifies the path and file name where you want to redirect standard input from.

Limits: Up to 255 characters

Notes:

- The job owner must have read permission to the specified file on the client computer.
- When specifying standard input, output, and error file locations, we recommend that you enter absolute paths to the files instead of relative paths or file names alone.
- You can use variables exported from the job profile or from global variables in the path name specification. Enclose variables referenced in the job profile in braces (for example, "\${PATH}") . Use the format \${global_name} for global variables.

Windows:

- If you do not specify a full path name, the path defaults to the job owner's home directory.

When specifying drive letters in job definitions, you must escape the colon with double quotation marks or backslashes. For example, C:\\tmp or "C:\\tmp" is valid; C:\\tmp is not.

JIL attribute: std_in_file

Standard output file

(Optional) Defines the path and file name where you want to redirect all standard output.

If you are running jobs across platforms, the scheduler of the issuing instance controls the default behavior. For UNIX, the default is to append this file. For Windows, the default is to overwrite this file. To overwrite the file, enter > as the first character in the value. To append the file, enter >> as the first characters in the value.

Limits: Up to 255 characters; it cannot contain spaces between the >> characters and the full path or file name

Notes:

- The job owner must have write permission to the specified file on the client computer.
- When specifying standard input, output, and error file locations, we recommend that you enter absolute paths to the files instead of relative paths or file names alone.
- You can use variables exported from the job profile or from global variables in the path name specification. Enclose variables referenced in the job profile in braces (for example, "\${PATH}") . Use the format \${global_name} for global variables.

UNIX:

- This value overrides the instance-wide setting for the AutoInstWideAppend parameter in the configuration file.
- This value overrides the machine-specific setting for the AutoMachWideAppend parameter in the /etc/auto.profile file.

Windows:

- If you do not specify a full path name, the path defaults to the job owner's home directory.
- This value overrides the instance-wide setting for the Append stdout/stderr field in the CA Workload Automation AE Administrator.
- When specifying drive letters in job definitions, you must escape the colon with double quotation marks or backslashes. For example, C:\\tmp or "C:\\tmp" is valid; C:\\tmp is not.

Default: /dev/null (UNIX) or NULL (Windows)

JIL attribute: std_out_file

Verify File Space Before a Job Starts

By default, jobs do not check the available file space. However, you can define a Command job to check if one or more file systems (UNIX) or drives (Windows) have the required amount of available space. At run time, the agent checks whether the required space is available on the client. 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 makes another attempt to verify the file space and start the job.

To verify file space before a job starts

1. Define a Command job.
2. Specify the following Command property in the job definition:

File check

Specifies the required amount of space on one or more file systems (UNIX) or drives (Windows) in kilobytes (KB).

3. Commit and run the job.

The file space is verified before the job starts.

Example: Verify the Available File Space on UNIX

This example checks if the file system named roots has 100 KB of available space. This example also checks if the file system named auxfs1 has 120 KB of available space. The specified file space must be available before the job can start.

To verify the available file space on UNIX

1. Create a Command job.
2. Enter the following properties:
 - Name—unix_chk
 - Send to machine—unixagent
 - Command—/u1/procrun.sh
 - File check—/roots 100 /auxfs1 120
3. Commit the job.

Example: Verify the Available File Space on Windows

This example checks if the C: drive has 100 KB of available space and if the D: drive has 120 KB of available space. The specified file space must be available before the job can start.

To verify the available file space on Windows

1. Create a Command job.
2. Enter the following properties:
 - Name—win_chk
 - Send to machine—winagent
 - Command—"C:\Programs\Payroll\pay.exe"
 - File check—"C: 100 D: 120"
3. Commit the job.

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.
2. Specify the Command property using the following format:

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 positional 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. Commit and run the job.

The positional arguments are passed to the program.

Example: Pass Positional Arguments to a UNIX Script

This example passes three arguments to the UNIX script addinfo.sh. The first argument passed is "user 1". This argument is enclosed with double quotation marks because it contains a space. The second argument passed is 905-555-1212, and the third argument is 749.

To pass positional arguments to a UNIX script

1. Create a Command job.
2. Enter the following properties:
 - Name—cmd_job1
 - Send to machine—unixprod
 - Command—addinfo.sh "user 1" 905-555-1212 749
3. Commit the job.

Example: Pass Positional Arguments to a Windows Program

This example passes two data files to the Windows program pay.exe. The arguments are enclosed with double quotation marks because they contain spaces.

To pass positional arguments to a Windows program

1. Create a Command job.
2. Enter the following properties:
 - Name—cmd_job2
 - Send to machine—winprod
 - Command—c:\Programs\Payroll\pay.exe "C:\Pay Data\salary.dat" "C:\Pay Data\benefits.dat"
3. Commit the job.

Example: Run cmd.exe

Suppose that you want the job COPYFILE to use the Windows command prompt (cmd.exe) to copy the file c:\env.txt to the test directory. To pass arguments to cmd.exe, you must enclose the argument in double quotes and precede the argument with the /C switch.

To run cmd.exe

1. Create a Command job.
2. Enter the following properties:
 - Name—COPYFILE
 - Send to machine—WINAGENT
 - Command—c:\Windows\system32\cmd.exe "/C copy c:\env.txt c:\test\env.txt"
3. Commit the job.

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 and Destinations

By default, the standard output and standard error output are redirected to /dev/null (UNIX) or NULL (Windows). You can define alternative error, input, and output sources and destinations in your job definition that override the defaults.

To define alternative error, input, and output sources and destinations

1. Define a Command job.
2. Add one or more of the following Command properties to the job definition:

Standard error file

Defines the path and file name where you want to redirect all standard error output.

Standard input file

Specifies the path and file name where you want to redirect standard input from.

Standard output file

Defines the path and file name where you want to redirect all standard output.

3. Commit and run the job.

The alternative sources are defined.

Chapter 7: User Defined Jobs

This section contains the following topics:

- [User Defined Jobs](#) (see page 109)
- [Define a User Defined Job](#) (see page 110)

User Defined Jobs

A User Defined job is similar to a Command job; however, the command is predefined on CA Workload Automation AE and not on a per-job basis. If your organization uses a command often in your workload definitions, you can predefined the command on CA Workload Automation AE and then make it available as a User Defined job type.

For example, suppose you often use the payroll.sh script to run UNIX workload. You can define the following job type 1 on CA Workload Automation AE:

```
insert_job: user1
job_type: 1
description:
command: /home/scripts/payroll.sh
machine: localhost
```

Once the job type 1 is defined, you can use it to create a User Defined job using Quick Edit or Application Editor. Within the User Defined job, you can add optional arguments to pass to the script. Jobs of type 1 execute /home/scripts/payroll.sh when those jobs are run. When a new version of the payroll.sh script is used, only the definition of job type 1 has to be modified.

Define a User Defined Job

You can define a User Defined job to run commands that are predefined on CA Workload Automation AE. You can define up to 9 User Defined jobs.

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

To define a User Defined job

1. Create a User Defined job in either Quick Edit or Application Editor.
The Properties section for the User Defined job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

3. (Optional) Specify optional [User Defined properties](#) (see page 110).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The User Defined job is defined.

User Defined Properties

The User Defined Job category includes the following properties:

Blob input

(Optional) Specifies the text to insert for the blob.

Limits: Up to 255 alphanumeric characters

JIL attribute: blob_input

Heartbeat interval

(Optional) Defines the frequency (in minutes) with which the job sends a heartbeat. The heartbeat is a signal that helps the scheduling manager monitor the progress of a job.

Default: 0 (The agent does not listen for heartbeats.)

Limits: 0-65535

JIL attribute: heartbeat_interval

Interactive

(Optional) Indicates whether the specified command needs to be user interactive.

JIL attribute: interactive

Job class

(Optional) Specifies the name of an initiator class defined in the agentparm.txt file. If the class name is not available on the target machine, the job will fail to be submitted.

Limits: Up to 32 characters

Note: The class must be a string without blanks in it.

JIL attribute: job_class

Optional arguments

(Optional) Specifies one or more numeric or alphabetic strings of data to pass to a script or command when the job runs. The arguments must be listed in the order expected in the script or command and separated by blank spaces.

Note: To pass an argument containing spaces, you must enclose its value in double quotes, for example, 362 "629 630" 748.

Shell

(Optional) Specifies the name of the shell used to execute the script or command file. You can specify any of the following UNIX shells:

- /bin/ksh (Korn shell)
- /bin/sh (Bourne shell)
- /bin/bash (Bourne again shell)
- /bin/csh (C shell)
- /usr/local/bin/perl (Perl shell)

JIL attribute: shell

Standard error file

(Optional) Defines the path and file name where you want to redirect all standard error output.

If you are running jobs across platforms, the scheduler of the issuing instance controls the default behavior. For UNIX, the default is to append this file. For Windows, the default is to overwrite this file. To overwrite the file, enter > as the first character in the value. To append the file, enter >> as the first characters in the value.

Limits: Up to 255 characters; it cannot contain spaces between the >> characters and the full path or file name

Notes:

- The job owner must have write permission to the specified file on the client computer.
- When specifying standard input, output, and error file locations, we recommend that you enter absolute paths to the files instead of relative paths or file names alone.
- You can use variables exported from the job profile or from global variables in the path name specification. Enclose variables referenced in the job profile in braces (for example, "\${PATH}") . Use the format \${global_name} for global variables.

UNIX:

- This value overrides the instance-wide setting for the AutoInstWideAppend parameter in the configuration file.
- This value overrides the machine-specific setting for the AutoMachWideAppend parameter in the /etc/auto.profile file.

Windows:

- If you do not specify a full path name, the path defaults to the job owner's home directory.
- This value overrides the instance-wide setting for the Append stdout/stderr field in the CA Workload Automation AE Administrator.
- When specifying drive letters in job definitions, you must escape the colon with double quotation marks or backslashes. For example, C:\\tmp or "C:\\tmp" is valid; C:\\tmp is not.
- If the Windows command specified in the job definition does not exist, the job does not run. The standard error file is not created and the job log indicates that the error file is not found.

Default: /dev/null (UNIX) or NULL (Windows)

JIL attribute: std_err_file

Standard input file

(Optional) Specifies the path and file name where you want to redirect standard input from.

Limits: Up to 255 characters

Notes:

- The job owner must have read permission to the specified file on the client computer.
- When specifying standard input, output, and error file locations, we recommend that you enter absolute paths to the files instead of relative paths or file names alone.
- You can use variables exported from the job profile or from global variables in the path name specification. Enclose variables referenced in the job profile in braces (for example, "\${PATH}") . Use the format \${global_name} for global variables.

Windows:

- If you do not specify a full path name, the path defaults to the job owner's home directory.

When specifying drive letters in job definitions, you must escape the colon with double quotation marks or backslashes. For example, C:\\tmp or "C:\\tmp" is valid; C:\\tmp is not.

JIL attribute: std_in_file

Standard output file

(Optional) Defines the path and file name where you want to redirect all standard output.

If you are running jobs across platforms, the scheduler of the issuing instance controls the default behavior. For UNIX, the default is to append this file. For Windows, the default is to overwrite this file. To overwrite the file, enter > as the first character in the value. To append the file, enter >> as the first characters in the value.

Limits: Up to 255 characters; it cannot contain spaces between the >> characters and the full path or file name

Notes:

- The job owner must have write permission to the specified file on the client computer.
- When specifying standard input, output, and error file locations, we recommend that you enter absolute paths to the files instead of relative paths or file names alone.
- You can use variables exported from the job profile or from global variables in the path name specification. Enclose variables referenced in the job profile in braces (for example, "\${PATH}") . Use the format \${global_name} for global variables.

UNIX:

- This value overrides the instance-wide setting for the AutoInstWideAppend parameter in the configuration file.
- This value overrides the machine-specific setting for the AutoMachWideAppend parameter in the /etc/auto.profile file.

Windows:

- If you do not specify a full path name, the path defaults to the job owner's home directory.
- This value overrides the instance-wide setting for the Append stdio/stderr field in the CA Workload Automation AE Administrator.
- When specifying drive letters in job definitions, you must escape the colon with double quotation marks or backslashes. For example, C:\\tmp or "C:\\tmp" is valid; C:\\tmp is not.

Default: /dev/null (UNIX) or NULL (Windows)

JIL attribute: std_out_file

Type

(Optional) Specifies the name of the User Defined job defined on CA Workload Automation AE.

JIL attribute: job_type

Chapter 8: File Watcher Jobs

This section contains the following topics:

[File Watcher Jobs](#) (see page 115)

[Define a File Watcher Job](#) (see page 116)

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.

Define a File Watcher Job

You can define a File Watcher job to start 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.

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

To define a File Watcher job

1. Create a File Watcher (FW) job in either Quick Edit or Application Editor.
The Properties section for the File Watcher job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

File(s) to watch

Specifies the path to and name of the file to monitor.

3. (Optional) Specify optional [File Watcher job properties](#) (see page 117).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The File Watcher job is defined.

File Watcher Job Examples

The following example describes a sample File Watcher job that monitors file activity.

Example: Creating a File Watcher Job

This example creates a File Watcher job named EOD_watch. The job watches every 60 seconds for a file named EodTransFile, located in the /tmp directory. When the watched file reaches the specified minimum size and does not change between check intervals, it is considered complete. When this occurs, the File Watcher job ends with a SUCCESS status.

To create a File Watcher job

1. Create a File Watcher job.
2. Enter the following properties:
 - Name—EOD_watch
 - Send to machine—SYSAGENT
 - File(s) to watch—/tmp/EodTransFile
 - Minimum file size—50000
 - Watch interval—60
3. Commit the job.

File Watcher Properties

The File Watcher Job category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

File(s) to watch

Specifies the path to and name of the files to monitor. To specify multiple files, use the asterisk (*) and question mark (?) wildcard characters. Use * for any number of characters and ? for any single character. Do not use the * and ? in the path.

Limits: Up to 255 characters; case-sensitive

UNIX Notes:

- You can specify variables exported from the profile script or global variables in the name.
- Wildcard characters are expanded according to the wildcard expansion rules of the Bourne Shell.

Windows Notes:

- You can specify system environment variables, job profile environment variables, and global variables in the name. If the variable is referenced in the job profile, we recommend that you enclose the variable in braces (for example, \${PATH}). Use the expression \$\$global_name} for global variables.
- When specifying drive letters in job definitions, you must escape the colon with quotation marks or backslashes. For example, C:\tmp or "C:\tmp" is valid; C:\tmp is not.

Example: /tmp/a*.data

JIL attribute: watch_file

Minimum file size

(Optional) Specifies the minimum file size (in bytes) that must be reached to cause this job to complete.

JIL attribute: watch_file_min_size

Watch interval

(Optional) Specifies the frequency (in seconds) the File Watcher job checks for the existence and size of the watched file.

Example: 30 (check every 30 seconds)

JIL attribute: watch_interval

Chapter 9: Entity Bean and Session Bean Jobs

This section contains the following topics:

- [Entity Bean Jobs](#) (see page 119)
- [Session Bean Jobs](#) (see page 120)
- [Define an Entity Bean Job](#) (see page 122)
- [Define a Session Bean Job](#) (see page 129)
- [Edit Parameters](#) (see page 134)

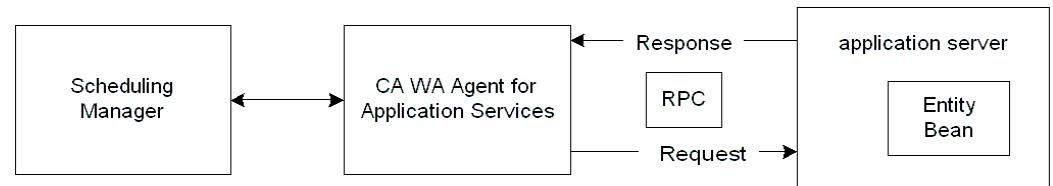
Entity Bean Jobs

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)

Session Bean Jobs

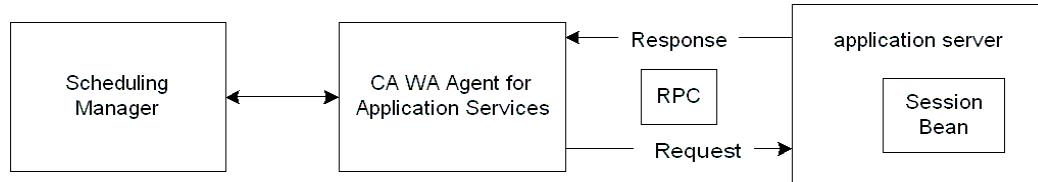
A session bean represents business logic or action to be taken (for example, charging a credit card or adding items to an online shopping cart).

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 in 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 an Entity Bean Job

You can define an Entity Bean 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. Create an Entity Bean (ENTYBEAN) job in either Quick Edit or Application Editor.

The Properties section for the Entity Bean job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

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.

Bean name

Specifies the JNDI name of the session or entity bean.

Operation type

Specifies the operation to perform on the entity bean. The following options are available:

- CREATE—Create an entity bean
- REMOVE—Remove an entity bean from the database
- UPDATE—Update the property values of an existing entity bean

Notes:

- To create an entity bean, you require the Create parameters properties.

- To update an entity bean, you require the Finder name, Finder parameters, Method name, and Modify parameter properties.
 - To remove an entity bean, you require the Finder name and Finder method properties.
3. (Optional) Specify optional [Entity Bean properties](#) (see page 125).
 4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
 5. Commit the job.

The Entity Bean job is defined.

Entity Bean Job Examples

The following examples describe sample Entity Bean jobs.

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.

To create an entity bean

1. Create an Entity Bean job.
2. Enter the following properties:
 - Name—newbean
 - Send to machine—APP_AGENT
 - Initial context factory—`weblogic.jndi.WLInitialContextFactory`
 - Provider URL—`t3://localhost:7001`
 - Bean name—customer
 - Operation type—CREATE
 - Create name—createcustomer
 - Create parameters—String=customerid, String=800-555-0100
3. Commit the job.

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.

To update an entity bean

1. Create an Entity Bean job.
2. Enter the following properties:
 - Name—upbean
 - Send to machine—APP_AGENT
 - Initial context factory—weblogic.jndi.WLInitialContextFactory
 - Provider URL—t3://localhost:7001
 - Bean name—customer
 - Operation type—UPDATE
 - Method name—changephone
 - Finder name—acme
 - Finder parameters—String=customerid, String=800-555-0199
3. Commit the job.

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.

To remove an entity bean

1. Create an Entity Bean job.
2. Enter the following properties:
 - Name—delbean
 - Send to machine—APP_AGENT
 - Initial context factory—weblogic.jndi.WLInitialContextFactory
 - Provider URL—t3://localhost:7001
 - Bean name—customer
 - Operation type—REMOVE
 - Method name—changephone
 - Finder name—acme
 - Finder parameters—String=customerid
3. Commit the job.

Entity Bean Properties

The Entity Bean category includes the following properties:

Agent job class

(Optional) Specifies the name of an initiator class defined in the agentparm.txt file. If the class name is not available on the target machine, the job will fail to be submitted.

Limits: Up to 32 characters

Note: The class must be a string without blanks in it.

Bean name

Specifies the JNDI name of the session or entity bean.

Limits: Up to 256 characters; case-sensitive

JIL attribute: bean_name

Create name

(Optional) Specifies the name of the create method.

Limits: Up to 256 characters

JIL attribute: create_name

Create parameters

(Optional) Specifies the [create parameters](#) (see page 134) to create an entity bean in a relational database on your application server.

Limits: 1024 for one J2EE parameter; up to 64 *type=value* pairs

Note: Specify the parameters in the same order as they appear in the method. The job will fail if the parameters are listed in the wrong order.

JIL attribute: create_parameter

Finder name

(Optional) Specifies the name of the finder method to update the property values of an existing entity bean or to 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.

Limits: Up to 256 characters; case-sensitive

JIL attribute: finder_name

Finder parameters

(Optional) Specifies the [finder parameters](#) (see page 134) to update the properties of an entity bean or remove an entity bean from a relational database on your application server.

Limits: 1024 for one J2EE parameter; up to 64 *type=value* pairs

Note: Specify the parameters in the same order as they appear in the method. The job fails if the parameters are listed in the wrong order.

JIL attribute: finder_parameter

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.

Limits: Up to 256 characters

JIL attribute: initial_context_factory

J2EE user name

(Optional) Specifies the JNDI user ID to be used to gain access to the connection factory.

Limits: Up to 145 characters; case-sensitive; cannot contain delimiters (such as spaces)

JIL attribute: j2ee_user

Method name

(Optional) Specifies the method to be invoked on the application server.

Limits: Up to 256 characters; case-sensitive

JIL attribute: method_name

Modify parameters

(Optional) Specifies the [modify parameters](#) (see page 134) to update the property values of an existing entity bean. The modify parameters are used by the setter method specified by the Method name property.

Limits: 1024 for one J2EE parameter; up to 64 *type=value* pairs

Note: Specify the parameters in the same order as they appear in the method. The job fails if the parameters are listed in the wrong order.

JIL attribute: modify_parameter

Operation type

Specifies the operation to perform on the entity bean. The following options are available:

- CREATE—Create an entity bean
- REMOVE—Remove an entity bean from the database
- UPDATE—Update the property values of an existing entity bean

JIL attribute: operation_type

Provider URL

Specifies the JNDI service provider URL. Both HTTP and HTTPS are supported. The following formats can be used:

- For WebLogic servers

`t3://WLIPAddress[:ORBPort]`

WLIPAddress

Specifies the IP address or domain name of the WebLogic Application Server.

ORBPort

(Optional) Specifies the ORB port.

Default: 7001

Example: `t3://localhost:7001`

- For WebSphere servers

`iiop://WSIPAddress[:ORBPort]`

WSIPAddress

Specifies the IP address or domain name of the WebSphere Application Server.

ORBPort

(Optional) Specifies the ORB port.

Default: 2809

Example: `iiop://172.24.0.0:2809`

Limits: Up to 256 characters; case-sensitive

JIL attribute: provider_url

Define a Session Bean Job

You can define a Session Bean 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. Create a Session Bean (SESSBEAN) job in either Quick Edit or Application Editor.

The Properties section for the Session Bean job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

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.

Bean name

Specifies the JNDI name of the session or entity bean.

Method name

Specifies the path to the servlet that should be invoked.

3. (Optional) Specify optional [Session Bean properties](#) (see page 131).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Session Bean job is defined.

Session Bean Job Examples

The following examples describe sample Session Bean jobs.

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 saves to the file C:\Makapt15. 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.

To invoke a method on a stateless session bean

1. Create a Session Bean job.
2. Enter the following properties:
 - Name—reverse
 - Send to machine—APP_AGENT
 - Initial context factory—com.ibm.websphere.naming.WsnInitialContextFactory
 - Provider URL—iiop://172.24.0.0:2809
 - Bean name—CybEJBTestBean
 - Method name—reverse
 - J2EE parameters—java.lang.String=a23
 - Destination file—C:\Makapt15
3. Commit the job.

Example: Invoke a Method on a Stateful Session Bean

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, when the Session Bean job runs, it adds two books to the shopping cart with ISBN numbers 1551929120 and 1582701709.

To invoke a method on a stateful session bean

1. Create a Session Bean job.
2. Enter the following properties:
 - Name—addbook
 - Send to machine—APP_AGENT
 - Initial context factory—com.ibm.websphere.naming.WsnInitialContextFactory
 - Provider URL—iiop://172.24.0.0:2809
 - Bean name—Shoppingcart
 - Method name—addbook
 - Create method—createaddbook
 - Create parameters—String=ISBN
 - J2EE parameters—integer=1551929120, integer=1582701709
3. Commit the job.

Session Bean Properties

The Session Bean category includes the following properties:

Agent job class

(Optional) Specifies the name of an initiator class defined in the agentparm.txt file. If the class name is not available on the target machine, the job will fail to be submitted.

Limits: Up to 32 characters

Note: The class must be a string without blanks in it.

Bean name

Specifies the JNDI name of the session or entity bean.

Limits: Up to 256 characters; case-sensitive

JIL attribute: bean_name

Create method

(Optional) Specifies the name of the create method.

Limits: Up to 256 characters; must always begin with create

JIL attribute: create_method

Create parameters

(Optional) Specifies the [create parameters](#) (see page 134) to create an entity bean in a relational database on your application server.

Limits: 1024 for one J2EE parameter; up to 64 *type=value* pairs

Note: Specify the parameters in the same order as they appear in the method. The job will fail if the parameters are listed in the wrong order.

JIL attribute: create_parameter

Destination file

(Optional) Specifies the output destination file to store the Java serialized object. If you omit this statement, the output is stored in the job's spool file or in the file referenced by the job's URI.

Limits: Up to 256 characters; case-sensitive

JIL attribute: destination_file

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.

Limits: Up to 256 characters

JIL attribute: initial_context_factory

J2EE parameters

(Optional) Specifies [input parameters](#) (see page 134) of the EJB Method to be invoked remotely.

Limits: 1024 for one J2EE parameter; up to 64 *type=value* pairs

Note: Specify the parameters in the same order as they appear in the method. The job will fail if the parameters are listed in the wrong order.

JIL attribute: j2ee_parameter

J2EE user name

(Optional) Specifies the JNDI user ID to be used to gain access to the connection factory.

Limits: Up to 145 characters; case-sensitive; cannot contain delimiters (such as spaces)

JIL attribute: j2ee_user

Method name

Specifies the method to be invoked on the application server.

Limits: Up to 256 characters; case-sensitive

JIL attribute: method_name

Provider URL

Specifies the JNDI service provider URL. Both HTTP and HTTPS are supported. The following formats can be used:

- For WebLogic servers

t3://WLIPAddress[:ORBPort]

WLIPAddress

Specifies the IP address or domain name of the WebLogic Application Server.

ORBPort

(Optional) Specifies the ORB port.

Default: 7001

Example: t3://localhost:7001

- For WebSphere servers

iiop://WSIPAddress[:ORBPort]

WSIPAddress

Specifies the IP address or domain name of the WebSphere Application Server.

ORBPort

(Optional) Specifies the ORB port.

Default: 2809

Example: iiop://172.24.0.0:2809

Limits: Up to 256 characters; case-sensitive

JIL attribute: provider_url

Edit Parameters

You can use the Edit Parameter dialog to define parameters for Entity Bean, Session Bean, JMS Publish, JMX-MBean Attribute Set, JMX-MBean Create Instance, POJO, and RMI, HTTP, and Web Service jobs.

To edit parameters

1. Open the Properties section for the job in either Quick Edit or Application Editor.

Note: For JMX-MBean Attribute Set jobs, skip to step 3.

2. Select Add at the end of the parameter field, and click Go.

A blank row appears below the property field to add a parameter.

3. Click  at the end of the row of the parameter you want to edit.

The Edit Parameter dialog opens.

4. Select one of the following options in the Parameter type field:

- Value—Identifies a single value
- Payload producing job—Identifies the name of a predecessor job
- Array—Identifies more than one value

Note: HTTP jobs only provide the Value option. Web Service jobs only provide the Value and Payload producing job options.

The Edit Parameter dialog displays new fields based on the chosen parameter type.

5. Complete the following fields as appropriate:

Array

Specifies a set of string values for the method parameter. Click Add to enter each value.

Note: This field is used with the Array parameter type.

Array type

Specifies the Java class of the parameter.

Note: This field is mandatory for the Value and Array parameter types only.

Payload producing job

Specifies the name of the predecessor job that produced the serialized Java object to be used as an input parameter.

Note: This field is used with the Payload producing job parameter type.

Value

Specifies the String value for the method parameter.

Note: This field is used with the Value parameter type.

Value type

Specifies the Java class of the parameter.

Note: This field is mandatory for the Value and Array parameter types only.

6. Click OK.

The parameter is added to the job definition.

Note: To make a change to the parameter, click  to reopen the Edit parameter dialog, edit the parameter, and click OK.

More information:

[Payload Producing Jobs](#) (see page 152)

Chapter 10: JMS Publish and JMS Subscribe Jobs

This section contains the following topics:

[JMS Publish and JMS Subscribe Jobs](#) (see page 137)

[Define a JMS Publish Job](#) (see page 141)

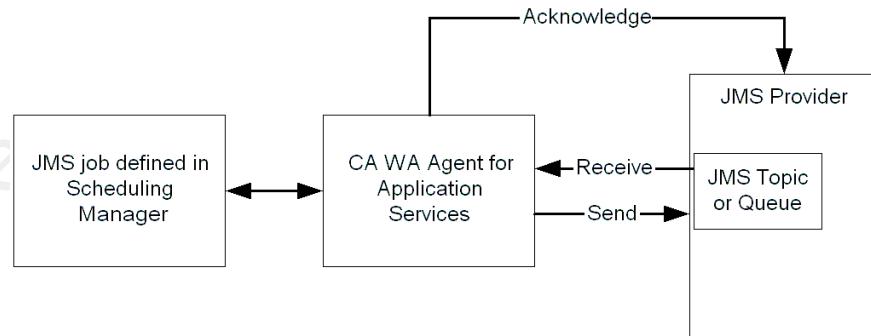
[Define a JMS Subscribe Job](#) (see page 146)

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.

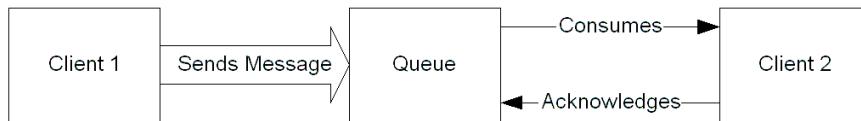
In a JMS system, a messaging server known as the JMS provider acts between a JMS client (the publisher) and another JMS client (the subscriber). Publishers send messages to the JMS provider while subscribers receive messages from the JMS provider.

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



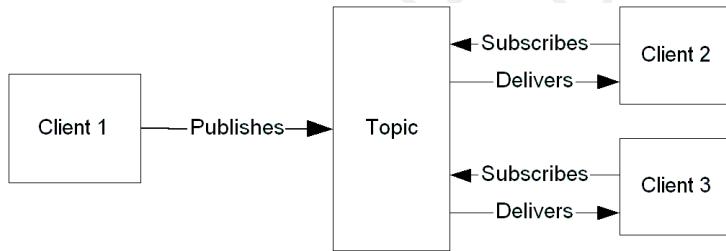
A queue is an object on the JMS server that holds messages sent by a client that are waiting to be consumed by another client. The queue retains a message until the message is consumed or the message expires.

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 a message arrives, the JMS provider is responsible for notifying all clients that messages have arrived for that topic.

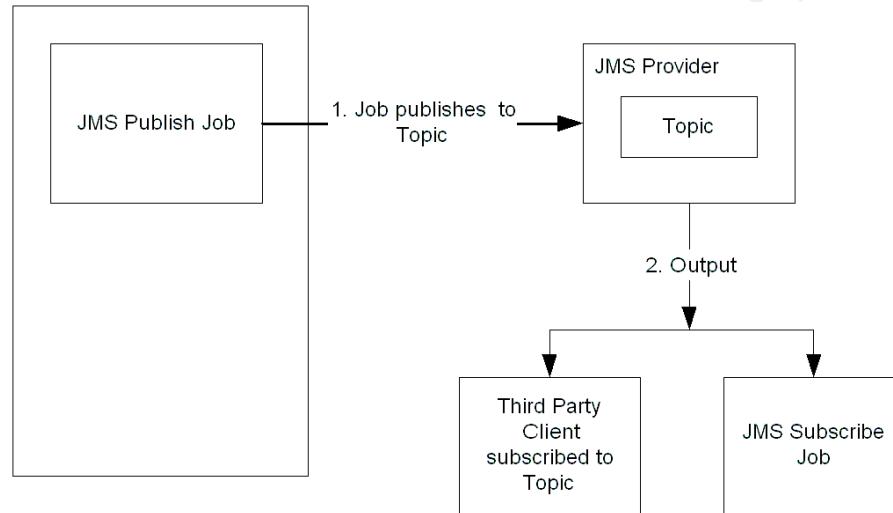
The following diagram shows two subscribers, Client 2 and Client 3, subscribed to a topic that the publisher, Client 1, publishes to:



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.

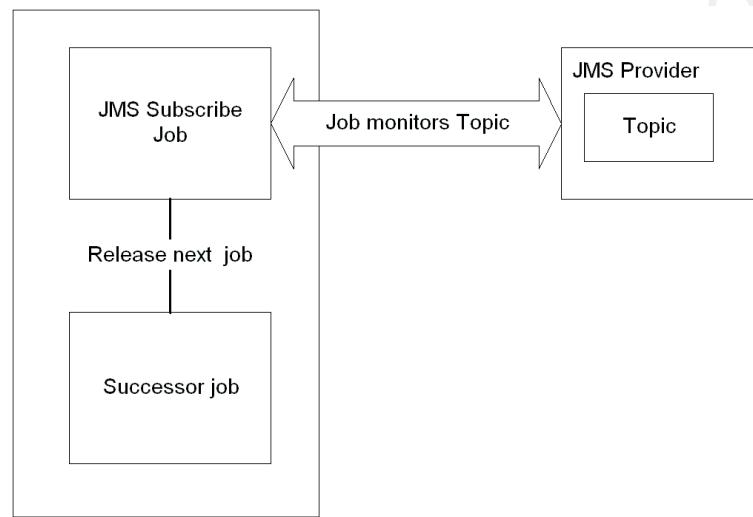
The following diagram shows a JMS Publish job scenario:



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. Create a JMS Publish (JMSPUB) job in either Quick Edit or Application Editor.
The Properties section for the JMS Publish job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

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.

Connection factory

Specifies the connection factory JNDI name that contains all the bindings needed to look up the referenced topic or queue.

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.

Message class

Specifies the Java class of the JMS message.

J2EE parameters

Specifies [input parameters](#) (see page 134) of the EJB Method to be invoked remotely.

3. (Optional) Specify optional [JMS Publish properties](#) (see page 143).

4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The JMS Publish job is defined.

JMS Publish Job Examples

The following example describes a sample JMS Publish job.

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 Application server and 2809 is the ORB port. The job uses the connection factory named ConnectionFactory.

To publish a message to the WebSphere Application Server

1. Create a JMS Publish job.
2. Enter the following properties:
 - Name—publish
 - Send to machine—APP_AGENT
 - Initial context factory—com.ibm.websphere.naming.WsnInitialContextFactory
 - Provider URL—iiop://172.24.0.0:2809
 - Connection factory—ConnectionFactory
 - Destination name—Queue
 - Use topic—unselected
 - Message class—String
 - J2EE parameters—java.lang.String='this is my message'
3. Commit the job.

JMS Publish Properties

The JMS Publish category includes the following properties:

Agent job class

(Optional) Specifies the name of an initiator class defined in the agentparm.txt file. If the class name is not available on the target machine, the job will fail to be submitted.

Limits: Up to 32 characters

Note: The class must be a string without blanks in it.

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.

Limits: Up to 256 characters; case-sensitive

JIL attribute: connection_factory

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.

Limits: Up to 256 characters; case-sensitive

JIL attribute: destination_name

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.

Limits: Up to 256 characters

JIL attribute: initial_context_factory

J2EE parameters

Specifies [input parameters](#) (see page 134) of the EJB Method to be invoked remotely.

Limits: 1024 for one J2EE parameter; up to 64 *type=value* pairs

Note: Specify the parameters in the same order as they appear in the method. The job will fail if the parameters are listed in the wrong order.

JIL attribute: j2ee_parameter

J2EE user name

(Optional) Specifies the JNDI user ID to be used to gain access to the connection factory.

Limits: Up to 145 characters; case-sensitive; cannot contain delimiters (such as spaces)

JIL attribute: j2ee_user

Message class

Specifies the Java class of the JMS message.

Limits: Up to 256 characters; case-sensitive

Note: If the package is not specified, java.lang is assumed.

JIL attribute: message_class

Provider URL

Specifies the JNDI service provider URL. Both HTTP and HTTPS are supported. The format is the following:

- For WebLogic servers

t3://WLIPAddress[:ORBPort]

WLIPAddress

Specifies the IP address or domain name of the WebLogic Application Server.

ORBPort

(Optional) Specifies the ORB port.

Default: 7001

Example: t3://localhost:7001

- For WebSphere servers

iiop://WSIPAddress[:ORBPort]

WSIPAddress

Specifies the IP address or domain name of the WebSphere Application Server.

ORBPort

(Optional) Specifies the ORB port.

Default: 2809

Example: iiop://172.24.0.0:2809

Limits: Up to 256 characters; case-sensitive

JIL attribute: provider_url

Use topic

(Optional) Indicates whether the job publishes messages to a topic or queue. When selected, the agent publishes messages to a topic.

Default: unselected (job publishes to a queue)

JIL attribute: use_topic

Define a JMS Subscribe Job

You can define a JMS Subscribe job to send a message 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. Create a JMS Subscribe (JMSSUB) job in either Quick Edit or Application Editor.
The Properties section for the JMS Subscribe job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

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.

Connection factory

Specifies the connection factory JNDI name that contains all the bindings needed to look up the referenced topic or queue.

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.

3. (Optional) Specify optional [JMS Subscribe properties](#) (see page 147).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The JMS Subscribe job is defined.

JMS Subscribe Job Examples

The following example describes a sample JMS Subscribe job.

Example: Monitor a Queue on a WebLogic Application Server

This example continuously monitors the queue named Queue (residing on WebLogic) for messages 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.

To monitor a queue on a WebLogic Application Server

1. Create a JMS Subscribe job.
2. Enter the following properties:
 - Name—monitor
 - Send to machine—APP_AGENT
 - Initial context factory—weblogic.jndi.WLInitialContextFactory
 - Provider URL—t3://172.24.0.0:7001
 - Connection factory—ConnectionFactory
 - Filter—abc\s...\\s[a-zA-Z]+\sFilter![\sa-zA-Z0-9]+
 - Destination file—/export/home/user1/outputfile1
 - Destination name—Queue
 - Use topic—unselected
3. Commit the job.

JMS Subscribe Properties

The JMS Subscribe category includes the following properties:

Agent job class

(Optional) Specifies the name of an initiator class defined in the agentparm.txt file. If the class name is not available on the target machine, the job will fail to be submitted.

Limits: Up to 32 characters

Note: The class must be a string without blanks in it.

JIL attribute: job_class

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.

Limits: Up to 256 characters; case-sensitive

JIL attribute: connection_factory

Continuous

(Optional) Indicates whether a job continuously monitors for a condition. When selected, the agent monitors for the conditions continuously and writes an alert to the scheduler log file each time a condition occurs.

Default: unselected (the job does not monitor for the conditions)

Note: To end continuous monitoring, you must complete the job manually.

JIL attribute: continuous

Destination file

(Optional) Specifies the output destination file that stores the consumed messages.

Limits: Up to 256 characters; case-sensitive

JIL attribute: destination_file

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.

Limits: Up to 256 characters; case-sensitive

JIL attribute: destination_name

Filter

(Optional) Specifies a regular expression to filter messages from the topic or queue. Any message not matching the filter is ignored. The scheduling manager uses the % character for JavaScript variables. To code a % character in a regular expression, use \% in your expression.

Limits: Up to 256 characters; case-sensitive

JIL attribute: filter

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.

Limits: Up to 256 characters

JIL attribute: initial_context_factory

J2EE user name

(Optional) Specifies the JNDI user ID to be used to gain access to the connection factory.

Limits: Up to 145 characters; case-sensitive; cannot contain delimiters (such as spaces)

JIL attribute: j2ee_user

Provider URL

Specifies the JNDI service provider URL. Both HTTP and HTTPS are supported. The following formats can be used:

- For WebLogic servers

`t3://WLIPAddress[:ORBPort]`

WLIPAddress

Specifies the IP address or domain name of the WebLogic Application Server.

ORBPort

(Optional) Specifies the ORB port.

Default: 7001

Example: `t3://localhost:7001`

- For WebSphere servers

`iiop://WSIPAddress[:ORBPort]`

WSIPAddress

Specifies the IP address or domain name of the WebSphere Application Server.

ORBPort

(Optional) Specifies the ORB port.

Default: 2809

Example: `iiop://172.24.0.0:2809`

Limits: Up to 256 characters; case-sensitive

JIL attribute: provider_url

Use topic

(Optional) Specifies whether the job publishes messages to a topic or queue. When selected, the agent publishes messages to a topic.

Default: unselected (the job publishes to a queue)

JIL attribute: use_topic

Chapter 11: JMX Jobs

This section contains the following topics:

- [JMX Jobs \(see page 151\)](#)
- [Payload Producing Jobs \(see page 152\)](#)
- [Define a JMX-MBean Attribute Get Job \(see page 154\)](#)
- [Define a JMX-MBean Attribute Set Job \(see page 157\)](#)
- [Define a JMX-MBean Create Instance Job \(see page 161\)](#)
- [Define a JMX-MBean Operation Job \(see page 164\)](#)
- [Define a JMX-MBean Remove Instance Job \(see page 168\)](#)
- [Define a JMX-MBean Subscribe Job \(see page 171\)](#)

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 the 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.

Payload Producing Jobs

A payload producing job is a job that produces binary output that is persisted as a serialized Java object or as a binary large object (BLOB). A BLOB is a collection of binary data stored in a relational database.

The following job types are payload producing jobs:

- HTTP
- JMX-MBean Attribute Get, JMX-MBean Attribute Set, JMX-MBean Operation
- POJO
- RMI
- Session Bean
- Web Service

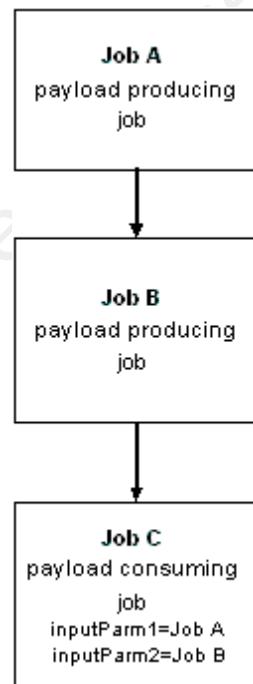
A payload consuming job describes a job that uses the output from a payload producing job as a parameter's input value. You can use the following job types as payload consuming jobs:

- Entity Bean
- JMS Publish
- JMX-MBean Attribute Set, JMX-MBean Operation
- POJO
- RMI
- Session Bean
- Web Service

The payload producing job must be a predecessor job to the payload consuming job, although it does not need to be an immediate predecessor.

Example: Payload producing job output as input to another job

The following diagram shows the relationship between three jobs. 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.



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. Create a JMX-MBean Attribute Get (JMXAG) job in either Quick Edit or Application Editor.

The Properties section for the JMX-MBean Attribute Get job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

URL

Specifies the URL to connect to the JMX server using an RMI connector.

MBean name

Specifies the full object name of an MBean.

Note: You can click  to open the Browse MBeans dialog to locate an MBean on the JMX server for the job definition.

MBean attribute

Specifies the name of the MBean attribute you want to query or set.

Note: You can click  to open the Browse MBean Attributes dialog to locate an MBean attribute on the JMX server for the job definition.

3. (Optional) Specify optional [JMX-MBean Attribute Get properties](#) (see page 155).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The JMX-MBean Attribute Get job is defined.

JMX-MBean Attribute Get Job Examples

The following example describes a sample JMX-MBean Attribute Get job.

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.

To query a JMX server for the value of an MBean attribute

1. Create a JMX-MBean Attribute Get job.
2. Enter the following properties:
 - Name—jmxa
 - Send to machine—APPAGENT
 - URL—service:jmx:rmi:///jndi/rmi://localhost:9999/server
 - MBean name—'DefaultDomain:index=1,type=Config'
 - MBean attribute—cachesize
3. Commit the job.

JMX-MBean Attribute Get Properties

The JMX-MBean Attribute Get category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

JMX user

(Optional) Specifies the user name.

Limits: Up to 145 characters

JIL attribute: jmx_user

MBean attribute

Specifies the name of the MBean attribute you want to query or set.

Limits: Up to 256 characters

JIL attribute: mbean_attr

MBean name

Specifies the full object name of an MBean. The format is as follows:

domain_name:key=value[,key=value...]

domain_name

Specifies the default domain name.

key=value[,key=value...]

Specifies one or more key value pairs.

Limits: Up to 256 characters; case-sensitive

Example: 'DefaultDomain:type=SimpleDynamic,index=3'

JIL attribute: mbean_name

Success pattern

(Optional) Specifies a regular expression to use as a filter.

Limits: Up to 256 characters; case-sensitive

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

JIL attribute: success_pattern

URL

Specifies the URL to connect to the JMX server using an RMI connector. The format is as follows:

service:jmx:rmi:///jndi/rmi://*hostName*:*portNum*/jmxrmi

hostName

Specifies the host name.

portNum

Identifies the port number specified when the agent was enabled.

Limits: Up to 256 characters; case-sensitive

Example: service:jmx:rmi:///jndi/rmi://localhost:9999/server

JIL attribute: url

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. Create a JMX-MBean Attribute Set (JMXAS) job in either Quick Edit or Application Editor.

The Properties section for the JMX-MBean Attribute Set job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

URL

Specifies the URL to connect to the JMX server using an RMI connector.

MBean name

Specifies the full object name of an MBean.

Note: You can click  to open the Browse MBeans dialog to locate an MBean on the JMX server for the job definition.

MBean attribute

Specifies the name of the MBean attribute you want to query or set.

Note: You can click  to open the Browse MBean Attributes dialog to locate an MBean attribute on the JMX server for the job definition.

JMX parameters

Specifies a [JMX parameter](#) (see page 134), as a type value pair, that the MBean attribute should be set to.

3. (Optional) Specify optional [JMX-MBean Attribute Set properties](#) (see page 158).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The JMX-MBean Attribute Set job is defined.

JMX-MBean Attribute Set Job Examples

The following example describes a sample JMX-MBean Attribute Set job.

Example: Change the Value of an MBean Attribute

Suppose that you want to set the value of the cachesize attribute of the Config MBean to the serialized Java object returned by a JMX-MBean Attribute Set job named size.

To change the value of an MBean attribute

1. Create a JMX-MBean Attribute Set job.
2. Enter the following properties:
 - Name—jmxa
 - Send to machine—APPAGENT
 - URL—service:jmx:rmi:///jndi/rmi://localhost:9999/server
 - MBean name—'DefaultDomain:index=1,type=Config'
 - MBean attribute—cachesize
 - JMX parameters—payload_job=size
3. Commit the job.

JMX-MBean Attribute Set Properties

The JMX-MBean Attribute Set category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

JMX parameters

Specifies a [JMX parameter](#) (see page 134), as a type value pair, that the MBean attribute should be set to.

Limits: Up to 1024 characters

JIL attribute: jmx_parameter

JMX user

(Optional) Specifies the user name.

Limits: Up to 145 characters

JIL attribute: jmx_user

MBean attribute

Specifies the name of the MBean attribute you want to query or set.

Limits: Up to 256 characters

JIL attribute: mbean_attr

MBean name

Specifies the full object name of an MBean. The format is as follows:

domain_name:key=value[,key=value...]

domain_name

Specifies the default domain name.

key=value[,key=value...]

Specifies one or more key value pairs.

Limits: Up to 256 characters; case-sensitive

Example: 'DefaultDomain:type=SimpleDynamic,index=3'

JIL attribute: mbean_name

Success pattern

Specifies a regular expression to use as a filter.

Limits: Up to 256 characters; case-sensitive

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

JIL attribute: success_pattern

URL

Specifies the URL to connect to the JMX server using an RMI connector. The format is as follows:

`service:jmx:rmi:///jndi/rmi://hostName:portNum/jmxrmi`

hostName

Specifies the host name.

portNum

Identifies the port number specified when the agent was enabled.

Limits: Up to 256 characters; case-sensitive

Example: `service:jmx:rmi:///jndi/rmi://localhost:9999/server`

JIL attribute: url

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. Create a JMX-MBean Create Instance (JMXMC) job in either Quick Edit or Application Editor.
The Properties section for the JMX-MBean Create Instance job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

URL

Specifies the URL to connect to the JMX server using an RMI connector.

MBean name

Specifies the full object name of an MBean.

Note: You can click  to open the Browse MBeans dialog to locate an MBean on the JMX server for the job definition.

Class name

Specifies fully-qualified Java class of the MBean object.

3. (Optional) Specify optional [JMX-MBean Create Instance properties](#) (see page 162).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The JMX-MBean Create Instance job is defined.

JMX-MBean Create Instance Job Examples

The following example describes a sample JMX-MBean Create Instance job.

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 class SimpleDynamic. The constructor of the class takes a single string parameter with the value "Hello".

To create an MBean instance on a JMX server

1. Create a JMX-MBean Create Instance job.
2. Enter the following properties:
 - Name—jmxmlc
 - Send to machine—APPAGENT
 - URL—service:jmx:rmi:///jndi/rmi://localhost:9999/server
 - MBean name—'DefaultDomain:type=SimpleDynamic,index=3'
 - Class name—SimpleDynamic
 - JMX parameters—java.lang.String=Hello
3. Commit the job.

JMX-MBean Create Instance Properties

The JMX-MBean Create Instance category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Class name

Specifies fully-qualified Java class of the MBean object.

Limits: Up to 1024 characters; case-sensitive; cannot contain delimiters (such as spaces)

JIL attribute: class_name

JMX parameters

(Optional) Specifies a [JMX parameter](#) (see page 134), as a type value pair, that the MBean attribute should be set to.

Limits: Up to 1024 characters

JIL attribute: jmx_parameter

JMX user

(Optional) Specifies the user name.

Limits: Up to 145 characters

JIL attribute: jmx_user

MBean name

Specifies the full object name of an MBean. The format is as follows:

domain_name:key=value[,key=value...]

domain_name

Specifies the default domain name.

key=value[,key=value...]

Specifies one or more key value pairs.

Limits: Up to 256 characters; case-sensitive

Example: 'DefaultDomain:type=SimpleDynamic,index=3'

JIL attribute: mbean_name

URL

Specifies the URL to connect to the JMX server using an RMI connector. The format is as follows:

service:jmx:rmi:///jndi/rmi://hostName:portNum/jmxrmi

hostName

Specifies the host name.

portNum

Identifies the port number specified when the agent was enabled.

Limits: Up to 256 characters; case-sensitive

Example: service:jmx:rmi:///jndi/rmi://localhost:9999/server

JIL attribute: url

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. Create a JMX-MBean Operation (JMXMOP) job in either Quick Edit or Application Editor.

The Properties section for the JMX-MBean Operation job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

URL

Specifies the URL to connect to the JMX server using an RMI connector.

MBean name

Specifies the full object name of an MBean.

Note: You can click  to open the Browse MBeans dialog to locate an MBean on the JMX server for the job definition.

MBean operation

Specifies the operation to be invoked.

Note: You can click  to open the Browse MBean Operations dialog to locate an operation for the job definition.

3. (Optional) Specify optional [JMX-MBean Operation properties](#) (see page 165).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The JMX-MBean Operation job is defined.

JMX-MBean Operation Job Examples

The following example describes a sample JMX-MBean Operation job.

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.

To invoke an operation on an MBean

1. Create a JMX-MBean Operation job.
2. Enter the following properties:
 - Name—jmxmlop
 - Send to machine—APPAGENT
 - URL—service:jmx:rmi:///jndi/rmi://localhost:9999/server
 - MBean name—'DefaultDomain:index=1,type=Config'
 - MBean operation—resetmem
 - JMX parameters—integer=50
3. Commit the job.

JMX-MBean Operation Properties

The JMX-MBean Operation category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

JMX parameters

(Optional) Specifies a [JMX parameter](#) (see page 134), as type value pair, of the MBean operation.

Limits: Up to 1024 characters

JIL attribute: jmx_parameter

JMX user

(Optional) Specifies the user name.

Limits: Up to 145 characters

JIL attribute: jmx_user

MBean name

Specifies the full object name of an MBean. The format is as follows:

domain_name:*key=value[,key=value...]*

domain_name

Specifies the default domain name.

key=value[,key=value...]

Specifies one or more key value pairs.

Limits: Up to 256 characters; case-sensitive

Example: 'DefaultDomain:type=SimpleDynamic,index=3'

JIL attribute: mbean_name

MBean operation

Specifies the name of the MBean operation to invoke.

Limits: Up to 256 characters; case-sensitive

JIL attribute: mbean_operation

Success pattern

(Optional) Specifies a regular expression to use as a filter.

Limits: Up to 256 characters; case-sensitive

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

JIL attribute: success_pattern

URL

Specifies the URL to connect to the JMX server using an RMI connector. The format is as follows:

`service:jmx:rmi:///jndi/rmi://hostName:portNum/jmxrmi`

hostName

Specifies the host name.

portNum

Specifies the port number specified when the agent was enabled.

Limits: Up to 256 characters; case-sensitive

Example: `service:jmx:rmi:///jndi/rmi://localhost:9999/server`

JIL attribute: url

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. Create a JMX-MBean Remove Instance (JMXREM) job in either Quick Edit or Application Editor.
2. The Properties section for the JMX-MBean Remove Instance job appears.

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

URL

Specifies the URL to connect to the JMX server using an RMI connector.

MBean name

Specifies the full object name of an MBean.

Note: You can click  to open the Browse MBeans dialog to locate an MBean on the JMX server for the job definition.

3. (Optional) Specify optional [JMX-MBean Remove Instance properties](#) (see page 169).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The JMX-MBean Remove Instance job is defined.

JMX-MBean Remove Instance Job Examples

The following example describes a sample JMX-MBean Remove Instance job.

Example: Remove an MBean Instance from a JMX Server

Suppose that you want to remove an MBean instance created by a JMX-MBean Create Instance job.

To remove an MBean instance from a JMX server

1. Create a JMX-MBean Remove Instance job.
2. Enter the following properties:
 - Name—jmxrem
 - Send to machine—APPAGENT
 - URL—service:jmx:rmi:///jndi/rmi://localhost:9999/server
 - MBean name—'DefaultDomain:type=SimpleDynamic,index=3'
3. Commit the job.

JMX-MBean Remove Instance Properties

The JMX-MBean Remove Instance category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

JMX user

(Optional) Specifies the user name.

Limits: Up to 145 characters

JIL attribute: jmx_user

MBean name

Specifies the full object name of an MBean. The format is as follows:

domain_name:key=value[,key=value...]

domain_name

Specifies the default domain name.

key=value[,key=value...]

Specifies one or more key value pairs.

Limits: Up to 256 characters; case-sensitive

Example: 'DefaultDomain:type=SimpleDynamic,index=3'

JIL attribute: mbean_name

URL

Specifies the URL to connect to the JMX server using an RMI connector. The format is as follows:

service:jmx:rmi:///jndi/rmi://hostName:portNum/jmxrmi

hostName

Specifies the host name.

portNum

Identifies the port number specified when the agent was enabled.

Limits: Up to 256 characters; case-sensitive

Example: service:jmx:rmi:///jndi/rmi://localhost:9999/server

JIL attribute: url

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. Create a JMX-MBean Subscribe (JMXSUB) job in either Quick Edit or Application Editor.

The Properties section for the JMX-MBean Subscribe job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

URL

Specifies the URL to connect to the JMX server using an RMI connector.

MBean name

Specifies the full object name of an MBean.

Note: You can click  to open the Browse MBeans dialog to locate an MBean on the JMX server for the job definition.

3. (Optional) Specify optional [JMX-MBean Subscribe properties](#) (see page 173).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The JMX-MBean Subscribe job is defined.

JMX-MBean Subscribe Job Examples

The following examples describe sample JMX-MBean Subscribe jobs.

Example: Set Up Notifications for Change to an MBean Attribute

Suppose that you want to set up continuous monitoring for changes to the cachesize attribute of the MBean named Config. The job filters the notifications the MBean sends by attribute. Each time the cachesize attribute changes, an alert is written to the scheduler log file.

To set up notifications for change to an MBean attribute

1. Create a JMX-MBean Subscribe job.
2. Enter the following properties:
 - Name—jmbsub1
 - Send to machine—APPAGENT
 - URL—service:jmx:rmi:///jndi/rmi://localhost:9999/server
 - MBean name—'DefaultDomain:index=1,type=Config'
 - Filter type—Attributes
 - Filter—cachesize
 - Continuous—selected
3. Commit the job.

Example: Set Up Notifications for Changes to Any MBean Attribute

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.

To set up notifications for changes to any MBean attribute

1. Create a JMX-MBean Subscribe job.
2. Enter the following properties:
 - Name—jmbsub2
 - Send to machine—APPAGENT
 - URL—service:jmx:rmi:///jndi/rmi://localhost:9999/server
 - MBean name—'DefaultDomain:index=1,type=Config'
 - Filter type—Attributes
 - Filter—jmx.attribute.change
 - Continuous—selected
3. Commit the job.

JMX-MBean Subscribe Properties

The JMX-MBean Subscribe category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Continuous

(Optional) Indicates whether a job continuously monitors for a condition. When selected, the agent monitors for the conditions continuously and writes an alert to the scheduler log file each time a condition occurs.

Default: unselected (the job does not monitor for the conditions)

Note: To end continuous monitoring, you must complete the job manually.

JIL attribute: continuous

Filter

(Optional) Specifies the name of attributes or the type of notifications to filter depending on the filter type specified.

Limits: Up to 256 characters

JIL attribute: filter

Filter type

(Optional) Specifies whether to filter notifications by attribute or by notification type, as follows:

- Attributes—Filters notifications by attribute.
- Types—Filters notifications by notification type.

Default: Attributes

JIL attribute: filter_type

JMX user

(Optional) Specifies the user name.

Limits: Up to 145 characters

JIL attribute: jmx_user

MBean name

Specifies the full object name of an MBean. The format is as follows:

domain_name:key=value[,key=value...]

domain_name

Specifies the default domain name.

key=value[,key=value...]

Specifies one or more key value pairs.

Limits: Up to 256 characters; case-sensitive

Example: 'DefaultDomain:type=SimpleDynamic,index=3'

JIL attribute: mbean_name

URL

Specifies the URL to connect to the JMX server using an RMI connector. The format is as follows:

`service:jmx:rmi:///jndi/rmi://hostName:portNum/jmxrmi`

hostName

Specifies the host name.

portNum

Identifies the port number specified when the agent was enabled.

Limits: Up to 256 characters; case-sensitive

Example: `service:jmx:rmi:///jndi/rmi://localhost:9999/server`

JIL attribute: url

Chapter 12: HTTP, POJO, RMI, and Web Service Jobs

This section contains the following topics:

- [HTTP Jobs](#) (see page 177)
- [POJO Jobs](#) (see page 178)
- [RMI Jobs](#) (see page 179)
- [Web Service Jobs](#) (see page 180)
- [Define an HTTP Job](#) (see page 181)
- [Define a POJO Job](#) (see page 188)
- [Define an RMI Job](#) (see page 191)
- [Define a Web Service Job Using Manual Entries](#) (see page 194)
- [Define a Web Service Job Using the Wizard](#) (see page 195)

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*.

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, speak to your agent administrator to verify the required JAR file is available in the jars subdirectory of the agent installation directory.

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.

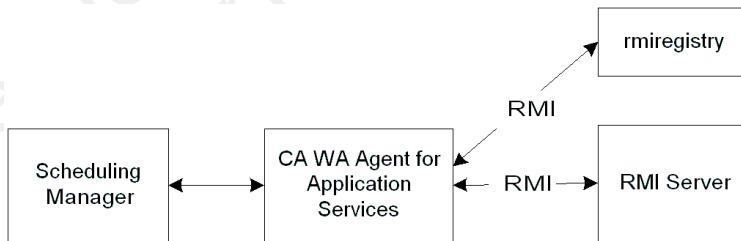
RMI jobs let you set up interaction between Java objects on different computers in a distributed network. Using an RMI job, you can access a remote server and invoke a method on a Java object. A method is a programmed procedure that is defined as a part of a Java class.

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:



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 the 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:



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 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. Create an HTTP job in either Quick Edit or Application Editor.

The Properties section for the HTTP job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Invocation type

Specifies the HTTP method type.

Provider URL

Specifies the host where the application to be invoked resides.

3. (Optional) Specify optional [HTTP properties](#) (see page 183).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The HTTP job is defined.

More information:

[HTTP Job Examples](#) (see page 182)

HTTP Job Examples

The following examples describe sample HTTP jobs.

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.

To define an HTTP job to subscribe to a mailing list

1. Create an HTTP job.
2. Enter the following properties:
 - Name—maillist
 - Send to machine—APP_AGENT
 - Invocation type—GET
 - Provider URL—`http://localhost:8080`
 - J2EE parameters—`string=subscribe, string=test@abc.com`
 - Method name—/examples/servlets/servlet/TheServlet
3. Commit the job.

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

This example sends a URL over HTTP using the GET method. The output of the invocation is returned in the job 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 user and BASIC, DIGEST, and NTLM protocols for web server authentication.

To define an HTTP job to send a URL over HTTP

1. Create an HTTP job.
2. Enter the following properties:
 - Name—URL
 - Send to machine—APP_AGENT
 - Invocation type—GET
 - Provider URL—http://host.example.com
 - J2EE connection origin—host.example.com
 - J2EE connection domain—windows_domain
 - J2EE authentication order—(BASIC, DIGEST, NTLM)
 - J2EE proxy host—proxy.example.com
 - J2EE proxy port—90
 - J2EE proxy user—user01 domain(mydomain)
 - J2EE proxy origin host—http://host.origin.proxy
 - J2EE proxy domain—http://host.domain.proxy
3. Commit the job.

HTTP Properties

The HTTP category includes the following properties:

Agent job class

(Optional) Specifies the name of an initiator class defined in the agentparm.txt file. If the class name is not available on the target machine, the job will fail to be submitted.

Limits: Up to 32 characters

Note: The class must be a string without blanks in it.

JIL attribute: job_class

Filter

(Optional) Specifies a regular expression to use as a filter. This property filters the output of the invoked servlet. Only those lines from the response that match the filter will be written to the spool file.

Limits: Up to 256 characters; case-sensitive

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

JIL attribute: filter

Invocation type

Specifies the HTTP method type, as follows:

GET

Sends the URL over HTTP using the GET method. The GET method requests data and sends the data as part of the URL.

POST

Sends the URL over HTTP using the POST method. The POST method submits data and is the preferred method for sending lengthy form data.

Default: POST

JIL attribute: invocation_type

J2EE authentication order

(Optional) Specifies a list of protocols to be used by a web server for authentication in an HTTP job. You can specify any of the following protocols in any order: BASIC, DIGEST, and NTLM. To specify more than one protocol, place the protocols within parentheses, separated by commas or blanks.

Limits: Up to 32 characters

Note: If you are connecting to a web server that cannot negotiate authentication protocols, enter the following list in the specified order: (BASIC, DIGEST, NTLM).

JIL attribute: j2ee_authentication_order

J2EE connection domain

(Optional) Specifies the domain for NTLM connection authentication. This property is required for NTLM authentication.

Limits: Up to 80 characters; case-sensitive

JIL attribute: j2ee_conn_domain

J2EE connection origin

(Optional) Specifies the origin host name for NTLM connection authentication. If you omit this property, the origin defaults to the computer name where the agent is running.

Limits: Up to 80 characters; case-sensitive

JIL attribute: j2ee_conn_origin

J2EE no global proxy defaults

(Optional) Indicates whether the agent uses the global proxy defaults specified in the agentparm.txt file. When selected, the agent ignores the global proxy configuration.

Default: selected (the agent ignores the global proxy configuration)

Note: You might select this option if the request is going to a server on the LAN that does not require the proxy.

JIL attribute: j2ee_no_global_proxy_defaults

J2EE parameters

(Optional) Specifies [input parameters](#) (see page 134) of the EJB Method to be invoked remotely.

Limits: 1024 for one J2EE parameter; up to 64 *type=value* pairs

Note: Specify the parameters in the same order as they appear in the method. The job will fail if the parameters are listed in the wrong order.

JIL attribute: j2ee_parameter

J2EE proxy domain

(Optional) Specifies the domain for proxy authentication required for NTLM authentication. If you omit this property, the domain defaults to the value specified in the http.proxyDomain parameter in the agentparm.txt file.

Limits: Up to 80 characters; case-sensitive

JIL attribute: j2ee_proxy_domain

J2EE proxy host

(Optional) Specifies the proxy host name to use for the request. If you omit this property, the host name defaults to the value specified in the http.proxyHost parameter in the agentparm.txt file.

Limits: Up to 256 characters; case-sensitive

JIL attribute: j2ee_proxy_host

J2EE proxy origin host

(Optional) Specifies the origin host name for proxy authentication used for NTLM authentication. If you omit this property, the host name defaults to the value specified in the http.proxyOrigin parameter in the agentparm.txt file.

Limits: Up to 256 characters; case-sensitive

JIL attribute: j2ee_proxy_origin_host

J2EE proxy port

(Optional) Specifies the port of the proxy server to use for the request. If you omit this property, the port defaults to the value specified in the http.proxyPort parameter in the agentparm.txt file.

Limits: 0-65535

JIL attribute: j2ee_proxy_port

J2EE proxy user

(Optional) Specifies the user name required for proxy authentication. If you omit this property, the user name defaults to the value specified in the http.proxyUser parameter in the agentparm.txt file.

Limits: Up to 80 characters; case-sensitive; cannot contain delimiters such as spaces

JIL attribute: j2ee_proxy_user

Method name

(Optional) Specifies the path to the servlet or query to be invoked.

Limits: Up to 256 characters; case-sensitive

JIL attribute: method_name

Provider URL

Specifies the host where the application to be invoked resides. The URL has the following format:

http://host:port/action

host

Specifies the name of the computer running the application server.

port

Specifies the port the host uses to listen for HTTP requests.

Default: 80

action

(Optional) Specifies the name of the program or servlet to be invoked. If you omit the action, then you must enter a value in the Method name field.

Limits: Up to 256 characters; case-sensitive

JIL attribute: provider_url

Define a POJO Job

You can define a POJO job to create a no-argument Java object instance, 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 CA WA Agent for Application Services.

To define a POJO job

1. Create a POJO job in either Quick Edit or Application Editor.

The Properties section for the POJO job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Class name

Specifies the name of the Java class to use.

Method name

Specifies the Java method to call.

3. (Optional) Specify optional [POJO properties](#) (see page 189).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The POJO job is defined.

More information:

[POJO Job Examples](#) (see page 189)

POJO Job Examples

The following example describes a sample POJO job.

Example: Invoke a Method on a Java Object Instance

Suppose that you want to define a POJO job that creates an empty Java string and calls the method parseInt with the argument "5". The job instantiates the java.lang.Integer class.

To invoke a method on a Java object instance

1. Create a POJO job.
2. Enter the following properties:
 - Name—parse
 - Send to machine—APP_AGENT
 - Class name— java.lang.Integer
 - Method name—parseInt
 - J2EE parameters—java.lang.String=5
3. Commit the job.

POJO Properties

The POJO category includes the following properties:

Agent job class

(Optional) Specifies the name of an initiator class defined in the agentparm.txt file. If the class name is not available on the target machine, the job will fail to be submitted.

Limits: Up to 32 characters

Note: The class must be a string without blanks in it.

Class name

Specifies the name of the Java class to use.

Limits: Up to 1024 characters; case-sensitive

JIL attribute: class_name

Destination file

(Optional) Specifies the output destination file.

Limits: Up to 256 characters; case-sensitive

JIL attribute: destination_file

J2EE parameters

(Optional) Specifies [input parameters](#) (see page 134) of the EJB Method to be invoked remotely.

Limits: 1024 for one J2EE parameter; up to 64 *type=value* pairs

Note: Specify the parameters in the same order as they appear in the method. The job will fail if the parameters are listed in the wrong order.

JIL attribute: j2ee_parameter

Method name

Specifies the Java method to call.

Limits: Up to 256 characters; case-sensitive

JIL attribute: method_name

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. Create an RMI job in either Quick Edit or Application Editor.

The Properties section for the RMI job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Remote name

Specifies the reference location of the object you want to invoke a method on.

Method name

Specifies a method name to execute on a remote object defined by the Remote name property.

3. (Optional) Specify optional [RMI properties](#) (see page 192).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The RMI job is defined.

More information:

[RMI Job Example](#) (see page 192)

RMI Job Example

The following example describes a sample RMI job.

Example: Define a Job to Start a Remote Server Immediately

Suppose that you want to invoke a method that starts a remote server. You want the server to start immediately.

To define a job to start a remote server immediately

1. Create an RMI job.
2. Enter the following properties:
 - Name—startsvr
 - Send to machine—APP_AGENT
 - Remote name—rmi://remotehost/Test
 - Method name—startserver
 - J2EE parameters—string=now
3. Commit the job.

RMI Properties

The RMI category includes the following properties:

Agent job class

(Optional) Specifies the name of an initiator class defined in the agentparm.txt file. If the class name is not available on the target machine, the job will fail to be submitted.

Limits: Up to 32 characters

Note: The class must be a string without blanks in it.

JIL attribute: job_class

Destination file

(Optional) Specifies the output file where the method execution output is stored. If you do not specify a file, the output will be written to the spool file by default.

Limits: Up to 256 characters; case-sensitive

JIL attribute: destination_file

J2EE parameters

(Optional) Specifies [input parameters](#) (see page 134) of the EJB Method to be invoked remotely.

Limits: 1024 for one J2EE parameter; up to 64 *type=value* pairs

Note: Specify the parameters in the same order as they appear in the method. The job will fail if the parameters are listed in the wrong order.

JIL attribute: j2ee_parameter

Method name

Specifies the method of the remote Java class to invoke.

Limits: Up to 256 characters; case-sensitive

JIL attribute: method_name

Remote name

Specifies the reference location of the object you want to invoke a method on. The format is the following:

rmi://[host:port/]name

host

(Optional) Specifies the name of the local or remote computer where the RMI registry that hosts the remote object runs.

Default: localhost

port

(Optional) Specifies the RMI registry port number the host uses to listen for requests.

Default: 1099, the default port of the host's RMI registry

name

Specifies the name of a remote object.

Limits: Up to 256 characters; case-sensitive

JIL attribute: remote_name

Define a Web Service Job Using Manual Entries

You can define a Web Service job to call an operation within a web service. The easiest way to define a Web Service job is to use the wizard. However, you may want to enter the job definition values manually. The target namespace and the name of the operation you want to invoke are required. Depending on the web service provider, other values may also be required.

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. Create a Web Service (WBSVC) job in either Quick Edit or Application Editor.
The Properties section for the Web Service job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Target namespace

Specifies the target namespace used for the names of messages, port type, binding, and service defined in the web service WSDL. Complex data types such as arrays require the target namespace.

Operation

Specifies the operation to be invoked.

3. (Optional) Specify optional [Web Service properties](#) (see page 198).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Web Service job is defined.

More information:

[Web Service Job Examples](#) (see page 196)

[Web Service Properties](#) (see page 198)

Define a Web Service Job Using the Wizard

The Web Service job features a wizard that lets you locate the WSDL document for a web service, select the operation, and enter the parameter values. You must use the wizard if you need to specify your company's proxy settings to access the Internet. Whenever possible, use the wizard to prevent entry errors.

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 using the wizard

1. Create a Web Service (WBSVC) job in either Quick Edit or Application Editor.
The Properties section for the Web Service job appears.
2. Click  in the WSDL location field to open the WSDL wizard.
The Browse WSDL dialog opens displaying "Step 1 of 5".
3. Complete the following fields, and click Next:

WSDL location

Indicates whether the WSDL file location is local or remote, as follows:

- Local WSDL File
- WSDL URL

Note: When you select WSDL URL, the wizard displays a Proxy section to specify a proxy server. If your company uses a proxy server to connect to the Internet, you must complete the Proxy fields.

WSDL

Specifies the local path or remote URL for the WSDL file.

Limits: Up to 256 characters

- "Step 2 of 5" appears.
4. Select a web service from the List of services and click Next.
"Step 3 of 5" appears.
 5. Select a port from the List of ports and click Next.
"Step 4 of 5" appears.
 6. Select an operation from the List of operation and click Next.
"Step 5 of 5" appears.
 7. Specify values for the input parameters of the operation.

8. Click Finish to save your input.

The Browse WSDL dialog closes and values for the web service appear in the job definition.

9. (Optional) Specify optional [Web Service properties](#) (see page 198).
(Optional) Specify [common properties that apply to all job types](#) (see page 39).

10. Commit the job.

The Web Service job is defined.

More information:

[Web Service Job Examples](#) (see page 196)

[Web Service Properties](#) (see page 198)

Web Service Job Examples

The following examples describe sample Web Service jobs.

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 the company is stored as a serialized Java object in the job's spool directory. In this example, the company is CA.

To get a company stock quote

1. Create a Web Service job.
2. Enter the following properties:
 - Name—quote
 - Send to machine—WSAGENT
 - WSDL
 - location—"http://www.webservicex.com/stockquote.asmx?WSDL"
 - Target namespace—"http://www.webserviceX.NET/"
 - Service name—StockQuote
 - Port name—StockQuoteSoap
 - Operation—GetQuote
 - End point URL—"http://www.webservicex.com/stockquote.asmx"
 - Operation parameters—xsd\string="CA"
 - Return class name—java.lang.String
 - Return xml type—string
 - Return namespace—"http://www.webserviceX.NET/"
 - One way—unselected
3. Commit the job.

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 operation IsValidEmail 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. In this example, you are validating John Smith's email address.

To validate an email address in a Web Service job

1. Create a Web Service job.
2. Enter the following properties:
 - Name—subscribe
 - Send to machine—WSAGENT
 - WSDL location—"http://www.webservicex.net/ValidateEmail.asmx?wsdl"
 - Target namespace—"http://www.webserviceX.NET/"
 - Service name—ValidateEmail
 - Port name—ValidateEmailSoap
 - Operation—IsValidEmail
 - End point URL—"http://www.webservicex.net/ValidateEmail.asmx"
 - Operation parameters—xsd\;string="john.smith@example.com"
 - Return class name—java.lang.Boolean
 - Return xml type—boolean
 - Return namespace—"http://www.webservicex.net"
 - Job criteria—true
3. Commit the job.

Web Service Properties

The Web Service category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Destination file

(Optional) Specifies the output destination file. In a payload producing job, the file stores the serialized Java object produced by a job.

Limits: Up to 256 characters; case-sensitive

JIL attribute: destination_file

End point URL

(Optional) Specifies the target endpoint address URL. In a published WSDL file, the URL defining the target endpoint address is found in the location attribute of the port's soap:address element.

Limits: Up to 256 characters; case-sensitive

JIL attribute: endpoint_url

Job criteria

(Optional) Specifies a regular expression to use as a filter.

Limits: Up to 256 characters; case-sensitive

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

JIL attribute: success_pattern

One way

(Optional) Specifies whether to invoke the web service one way. When selected, the agent invokes the service one way and the Web Service job completes without waiting for a response.

JIL attribute: one_way

Operation

Specifies the operation to be invoked.

Limits: Up to 256 characters; case-sensitive

JIL attribute: wsdl_operation

Operation parameters

(Optional) Specifies the type=value pairs of [parameters](#) (see page 134) that are passed to the operation invoked by the Web Service job.

Limits: Each parameter can be up to 1024 characters

Note: You can specify up to 64 parameters. Order is important.

JIL attribute: web_parameter

Port name

(Optional) Specifies the WSDL port name within the target namespace. A WSDL port describes the operations exposed by a web service and defines the connection point to the web service.

Limits: Up to 100 characters; case-sensitive

JIL attribute: port_name

Return class name

(Optional) Specifies the Java class name of the return value.

Limits: Up to 1024 characters; case-sensitive

JIL attribute: return_class_name

Return namespace

(Optional) Specifies the XML namespace for the XML type that maps to the Java class name of the return value.

Limits: Up to 256 characters; case-sensitive

JIL attribute: return_namespace

Return XML type

(Optional) Specifies the XML type that maps to the Java class name of the return value.

Limits: Up to 256 characters; case-sensitive

JIL attribute: return_xml_name

Service name

(Optional) Specifies the web service name within the target namespace.

Limits: Up to 255 characters; case-sensitive

JIL attribute: service_name

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.

Limits: Up to 256 characters; case-sensitive

JIL attribute: target_namespace

User

(Optional) Specifies the user name.

Limits: Up to 145 characters

JIL attribute: web_user

WSDL location

Specifies the local path or remote URL for the WSDL file.

Click  to open a wizard.

Limits: Up to 256 characters; case-sensitive

JIL attribute: wsdl_url

Chapter 13: Database Jobs

This section contains the following topics:

- [Database Jobs](#) (see page 203)
- [How Database Trigger Jobs Differ from Database Monitor Jobs](#) (see page 204)
- [Define a Database Monitor Job](#) (see page 205)
- [Define a Database Stored Procedure Job](#) (see page 209)
- [Define a Database Trigger Job](#) (see page 219)
- [Define an SQL Job](#) (see page 231)

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:

Database Monitor

Lets you monitor for an increase or decrease in the number of rows in a database table.

Database Stored Procedure

Lets you run a stored procedure.

Database Trigger

Lets you monitor for added, deleted, and updated rows in a database table.

SQL

Lets you execute an SQL statement.

How Database Trigger Jobs Differ from Database Monitor Jobs

You can monitor database tables using Database Monitor or Database Trigger jobs.

A Database Monitor job can monitor a database table for an increase or decrease in the number of rows. To monitor the database for specific changes, you can add a monitor condition to the job definition. Database Monitor jobs count the number of rows that satisfy the monitor condition using a polling interval, which is every 10 seconds by default. Database Monitor jobs do not detect other updates to a row or changes to the number of rows that cancel each other out during the polling interval. For example, suppose that within a 10-second interval, a row is added while another row is deleted. Since the total number of rows did not change, the Database Monitor job does not detect the row addition and deletion.

Alternatively, a Database Trigger job can monitor a database table for added rows, deleted rows, or updated rows. To monitor the database for specific changes, you can add a trigger condition to the job definition. Each Database Trigger job creates a database trigger on the database. The database trigger templates that the agent uses are located in the directory where the agent is installed. The template files are named dbtrigDBtype.properties, for example, dbtrigOracle.properties. Contact your database administrator before choosing to use a Database Trigger job.

For either job type, you can set up continuous monitoring so that each time a database change occurs, an alert or an event is triggered.

Note: Not all scheduling managers support alerts and events.

Important! You should not drop a table that is being referenced by either a Database Monitor or a Database Trigger job. The job will remain active, even though the table has been dropped.

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. Create a Database Monitor (DBMON) job in either Quick Edit or Application Editor.

The Properties section for the Database Monitor job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

Table name

Specifies the name of the database table to monitor for changes.

Database connect string

Specifies a JDBC database resource location (URL).

Note: Your agent administrator can also set a default value for this property using the db.default.url parameter in the agentparm.txt file.

3. (Optional) Specify optional [Database Monitor properties](#) (see page 207).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Database Monitor job is defined.

More information:

[Database Monitor Job Examples](#) (see page 206)

Database Monitor Job Examples

The following examples describe sample Database Monitor jobs.

Example: Monitor a Table for a Change in the Number of Rows

Suppose that you want a job to monitor the STAFF table for a change in the number of rows. When a row that has the name Jonson is added or deleted, the job completes. The job uses the database resource location default defined on the agent.

To monitor a table for a change in the number of rows

1. Create a Database Monitor job.
2. Enter the following properties:
 - Name—staff
 - Send to machine—DB_AGENT
 - Owner—entadm
 - Table name—staff
 - Monitor type—VARIANCE
 - Monitor condition—Jonson
3. Commit the job.

Example: Monitor a Table for Increases in the Number of Rows

Suppose that you want a job to monitor the EMP table for increases in the number of rows. When a new row has a salary greater than 100000, the scheduling manager sends an alert. The job remains in a monitoring state until it is forced complete or cancelled. The job uses the user name and database resource location defaults defined on the agent.

To monitor a table for increases in the number of rows

1. Create a Database Monitor job.
2. Enter the following properties:
 - Name—emp
 - Send to machine—DB_AGENT
 - Table name—emp
 - Monitor type—INCREASE
 - Monitor condition—select count(*) from emp where sal>100000
3. Commit the job.

Database Monitor Properties

The Database Monitor category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Continuous

(Optional) Indicates whether to monitor for the conditions continuously. When selected, each time the specified conditions occur, an alert is written to the scheduler log file.

Default: unselected (the job does not monitor for the conditions)

Note: To end continuous monitoring, you must complete the job manually.

JIL attribute: continuous

Database connect string

Specifies a JDBC database resource location (URL) for a job. Use the appropriate format for your database type, as follows:

- For an Oracle database:

"jdbc:oracle:thin:@host:port:dbname"

- For a Microsoft SQL Server database:

"jdbc:sqlserver://host:port;DatabaseName=dbname"

- For an IBM DB2 database:

"jdbc:db2://host:port/dbname"

Limits: Up to 256 characters; case-sensitive

JIL attribute: connect_string

Monitor condition

(Optional) Specifies the condition to monitor in the database. This condition is equivalent to an SQL where clause.

Limits: Up to 500 characters; case-sensitive

JIL attribute: monitor_cond

Monitor type

(Defaulted) Specifies the type of database change to monitor for, as follows:

DECREASE (Row decrease)

Monitors for a decrease in the number of rows in the database table.

INCREASE (Row increase)

Monitors for an increase in the number of rows in the database table.

VARIANCE (Both of the above)

Monitors for an increase or a decrease in the number of rows in the database table.

Default: VARIANCE (Both of the above)

JIL attribute: monitor_type

Oracle user role

(Optional) Specifies the role of the Oracle user to log in as. The user role must be defined in the Oracle database. The user role uses the following syntax:

as userrole

Limits: Up to 64 characters; case-sensitive

Example: as sysdba

JIL attribute: user_role

Table name

Specifies the name of the database table to monitor for changes.

Limits: Up to 300 characters

JIL attribute: tablename

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 the scheduling manager. 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. Create a Database Stored Procedure (DBPROC) job in either Quick Edit or Application Editor.

The Properties section for the Database Stored Procedure job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

Stored procedure

Specifies the name of the database stored procedure to run.

Database connect string

Specifies a JDBC database resource location (URL).

Note: Your agent administrator can also set a default value for this property using the db.default.url parameter in the agentparm.txt file.

3. (Optional) Specify optional [Database Stored Procedure properties](#) (see page 214).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Database Stored Procedure job is defined.

More information:

[Supported Data Types](#) (see page 217)

[Database Stored Procedure Job Examples](#) (see page 210)

Database Stored Procedure Job Examples

The following examples describe sample Database Stored Procedure jobs.

Example: Invoke a Simple Stored Procedure from a Microsoft SQL Server Database

Suppose that you want a job to run the byroyalty stored procedure located in the pubs sample database. When the job runs, the agent passes the input parameter percentage a value of 40.

To invoke a simple stored procedure from a Microsoft SQL Server database

1. Create a Database Stored Procedure job.
2. Enter the following properties:
 - Name—staff
 - Send to machine—DB_AGENT
 - Owner—sa
 - Stored procedure—byroyalty
 - Parameters list:

Argtype—IN
Name—percentage
Datatype—INTEGER
Value—40
Ignore—No
 - Database connect string—"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
3. Commit the job.

Example: Invoke a Stored Procedure with Input and Output Parameters from a Microsoft SQL Server Database

Suppose that you want a job to run the emp stored procedure, which returns a value from the emp table. The pubid returned matches the employee named Ann Devon. The pubid, 9952, appears in the job's spool file.

To invoke a stored procedure with input and output parameters from a Microsoft SQL Server database

1. Create a Database Stored Procedure job.
2. Enter the following properties:
 - Name—staff
 - Send to machine—DB_AGENT
 - Owner—sa
 - Stored procedure—emp
 - Parameters list:
 - Argtype—IN
Name—f_name
Datatype—VARCHAR
Value—Ann
 - Parameters list:
 - Argtype—IN
Name—l_name
Datatype—VARCHAR
Value—Devon
 - Parameters list:
 - Argtype—OUT
Name—pubid
Datatype—CHAR
 - Database connect
string—"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
 - 3. Commit the job.

To run the job in this example, create the emp stored procedure in the pubs database using the following script:

```
CREATE PROCEDURE EMP
(@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
```

Example: Invoke a Stored Procedure with Input and Output Parameters from an IBM DB2 Database

Suppose that you want a job to run the DEPT_MEDIAN stored procedure under the user entadm. DEPT_MEDIAN returns the median salary of department 20 from the STAFF table. The median salary, 18171.25, appears in the job's spool file.

To invoke a stored procedure with input and output parameters from an IBM DB2 database

1. Create a Database Stored Procedure job.
2. Enter the following properties:
 - Name—staff
 - Send to machine—DB_AGENT
 - Owner—entadm
 - Stored procedure—ENTADM.DEPT_MEDIAN
 - Parameters list:
 - Arctype—IN
Name—deptNumber
Datatype—SMALLINT
Value—20
 - Parameters list:
 - Arctype—OUT
Name—medianSalary
Datatype—DOUBLE
 - Database connect string—"jdbc:db2://10.1.4.131:50000/SAMPLE"
 - 3. Commit the job.

The spool file for this job contains the following output:

```
{ call ENTADM.DEPT_MEDIAN(?, ?) }
medianSalary=18171.25
```

The job in this example runs the following stored procedure in the SAMPLE database:

```
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
```

Database Stored Procedure Properties

The Database Stored Procedure category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Database connect string

(Optional) Specifies a JDBC database resource location (URL) for a job. Use the appropriate format for your database type, as follows:

- For an Oracle database:

"jdbc:oracle:thin:@host:port:dbname"

- For a Microsoft SQL Server database:

"jdbc:sqlserver://host:port;DatabaseName=dbname"

- For an IBM DB2 database:

"jdbc:db2://host:port/dbname"

Limits: Up to 256 characters; case-sensitive

JIL attribute: connect_string

Oracle user role

(Optional) Specifies the role of the Oracle user to log in as. The user role must be defined in the Oracle database. The user role uses the following syntax:

as *userrole*

Limits: Up to 64 characters; case-sensitive

Example: as sysdba

JIL attribute: user_role

Parameters list: Argtype

(Optional) Specifies the parameter type, as follows:

- IN—Specifies that the parameter is an input parameter.
- OUT—Specifies that the parameter is an output parameter.
- INOUT—Specifies that the parameter is an input-output parameter.

JIL attribute: sp_arg: argtype

Parameters list: Datatype

(Optional) Specifies the database [data type](#) (see page 217) of the parameter.

Limits: Up to 100 characters; case-sensitive

JIL attribute: sp_arg: datatype

Parameters list: Ignore

(Optional) Specifies whether the value is ignored, as follows:

- Yes—Specifies that the value is not ignored.
- No—Specifies that the value is returned.

This property is useful if you want to hide data such as a password.

Default: No

Note: This argument applies to output parameters only.

JIL attribute: sp_arg: ignore

Parameters list: Name

(Optional) Specifies the name of the parameter.

Limits: Up to 64 characters

JIL attribute: sp_arg: name

Parameters list: Value

(Optional) Specifies the value of the parameter.

Limits: Up to 256 characters; case-sensitive

Notes:

- This argument applies to input parameters only.
- Enclose values that contain spaces in single quotation marks.

JIL attribute: sp_arg: value

Result type

(Optional) Specifies the data type to return from a stored procedure.

Limits: Up to 256 characters; case-sensitive

JIL attribute: result_type

Stored procedure

Specifies the database stored procedure to run.

Limits: Up to 300 characters

JIL attribute: sp_name

Success criteria

(Optional) Defines a regular expression that is used to evaluate a return string. If the return string matches the regular expression, the job completes successfully. Otherwise, the job fails.

Limits: Up to 500 characters; case-sensitive

Note: Each return string includes the field name from the SELECT statement and its value, separated by an equal sign (=). For example, consider the query SELECT ORD_NUM FROM SALES. To match order number A2976, specify the regular expression ORD_NUM=A2976. Specifying the regular expression A2976 does not match any return string causing the job to fail.

JIL attribute: success_criteria

Supported Data Types

The following table lists the supported data types of different databases. The data types listed in the database columns are defined in the procedure definition within the database. In the job definition, use the corresponding JDBC data type from the first column.

| JDBC Data Type | Valid Input Format | Oracle | MS SQL Server | DB2 |
|----------------|--------------------|---------------------------|----------------|---------------|
| CHAR | Plain text | CHAR | CHAR | CHAR |
| VARCHAR | Plain text | VARCHAR2(x) VARCHAR(x) | VARCHAR(x) | VARCHAR(x) |
| LONGVARCHAR | | Not supported | Not supported | Not supported |
| NUMERIC | Number | NUMBER, NUMERIC | NUMERIC | NUMERIC |
| DECIMAL | Number | DECIMAL | DECIMAL | DECIMAL |
| BIT | | Not supported | BIT | Not supported |
| BOOLEAN | | Not supported | Not supported | Not supported |
| TINYINT | | Not supported | TINYINT | Not supported |
| SMALLINT | Number | SMALLINT | SMALLINT | SMALLINT |
| INTEGER | Number | INTEGER | INTEGER INT | INTEGER |
| BIGINT | Number | Not supported | BIGINT | BIGINT |
| REAL | Number | REAL | REAL | REAL |

| JDBC Data Type | Valid Input Format | Oracle | MS SQL Server | DB2 |
|----------------|--|-----------|---------------------------|-----------|
| [dot Number] | | | | |
| FLOAT | Number [dot Number] | FLOAT | FLOAT | FLOAT |
| DOUBLE | | | | |
| DATE | yyyy-mm-dd String (e.g., SYSDATE) | DATE | Not supported | DATE |
| TIME | hh:mm:ss | | Not supported | TIME |
| TIMESTAMP | yyyy-mm-dd hh:mm:ss.ffff ffff | TIMESTAMP | datetime smalldatetime | TIMESTAMP |
| BINARY | Hex (e.g., 010203FF) | | timestamp | |

Note: Although JDBC provides functionality for Booleans, the Oracle database driver does not. To handle unsupported data types such as Booleans, you can call a wrapper procedure in the job definition. The wrapper procedure converts the input values to values supported by the database driver and calls the appropriate stored procedure from inside the database.

Example: Run a Wrapper Procedure

In the following example, the agent runs a wrapper procedure called boolwrap, which converts the inputted integer value to a Boolean and calls the boolproc stored procedure:

```
CREATE OR REPLACE PROCEDURE boolproc(x boolean) AS BEGIN
  [...]
END;
CREATE OR REPLACE PROCEDURE boolwrap(x int) AS BEGIN
  IF (x=1) THEN
    boolproc(TRUE);
  ELSE
    boolproc(FALSE);
  END IF;
END;
```

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. Create a Database Trigger (DBTRIG) job in either Quick Edit or Application Editor.

The Properties section for the Database Trigger job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

Table name

Specifies the name of the database table to monitor for changes.

Database type

Specifies whether the database is an Oracle, MSSQL, or DB2 database.

Trigger type

Specifies the type of database change to monitor for.

Database connect string

Specifies a JDBC database resource location (URL).

Note: Your agent administrator can also set a default value for this property using the db.default.url parameter in the agentparm.txt file.

3. (Optional) Specify optional [Database Trigger properties](#) (see page 228).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Database Trigger job is defined.

More information:

[Database Trigger Job Examples for Oracle](#) (see page 220)

[Database Trigger Job Examples for Microsoft SQL Server](#) (see page 224)

[Database Trigger Job Examples for IBM DB2](#) (see page 227)

Database Trigger Job Examples for Oracle

The following examples describe sample Database Trigger jobs that monitor Oracle database tables.

Example: Monitor an Oracle Database Table for an Added or Deleted Row

Suppose that you want a job to monitor 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. 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.

To monitor an Oracle database table for an added or deleted row

1. Create a Database Trigger job.
2. Enter the following properties:
 - Name—adddelrow
 - Send to machine—DB_AGENT
 - Table name—emp
 - Trigger type—INSERT,DELETE
 - Database connect string—"jdbc:oracle:thin:@myhost:1521:orcl"
 - Oracle user role—scott
3. Commit the job.

Example: Monitor an Oracle Database Table for Deleted Rows with a Trigger Condition

Suppose that you want a job to monitor the emp table for deleted rows. The job runs under a user who has the authority to create triggers on the database. When a row containing department 75 is deleted, the job completes.

To monitor an Oracle database table for deleted rows with a trigger condition

1. Create a Database Trigger job.
2. Enter the following properties:
 - Name—delrow
 - Send to machine—DB_AGENT
 - Table name—emp
 - Trigger type—DELETE
 - Trigger conditions—old.deptno=75
 - Database connect string—"jdbc:oracle:thin:@myhost:1521:orcl"
 - Oracle user role—scott
3. Commit the job.

Example: Monitor an Oracle Database Table for Added Rows with a Trigger Condition

Suppose that you want a job to monitor the emp table for added rows. The job runs under a user who has the authority to create triggers on the database. When a row containing a name beginning with the letter g is added, the job completes.

To monitor an Oracle database table for added rows with a trigger condition

1. Create a Database Trigger job.
2. Enter the following properties:
 - Name—addrow
 - Send to machine—DB_AGENT
 - Table name—emp
 - Trigger type—INSERT
 - Trigger conditions—'new.ename like "g%%"'
 - Database connect string—jdbc:oracle:thin:@myhost:1521:orcl
 - Oracle user role—scott
3. Commit the job.

Example: Monitor an Oracle Database Table for Added or Updated Rows with a Trigger Condition

Suppose that you want a job to monitor the emp table for added or updated rows. The job runs under a user who has the authority to create triggers on the database. The job completes when a new or updated row does not contain a job field equal to sales.

To monitor an Oracle database table for added or updated rows with a trigger condition

1. Create a Database Trigger job.
2. Enter the following properties:
 - Name—adduprow
 - Send to machine—DB_AGENT
 - Table name—emp
 - Trigger type—INSERT,UPDATE
 - Trigger conditions—new.job<>'sales'
 - Database connect string—"jdbc:oracle:thin:@myhost:1521:orcl"
 - Oracle user role—scott
3. Commit the job.

Note: The <> symbol indicates not equal to.

Database Trigger Job Examples for Microsoft SQL Server

The following examples describe sample Database Trigger jobs that monitor Microsoft SQL Server database tables.

Example: Monitor a Microsoft SQL Server Database Table for a New or Deleted Row

Suppose that you want a job to monitor the stores table for an added row or a deleted row. The job runs on the DB_AGENT agent computer under the sa user. 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.

To monitor a Microsoft SQL Server database table for a new or deleted row

1. Create a Database Trigger job.
2. Enter the following properties:
 - Name—newdelrow
 - Send to machine—DB_AGENT
 - Owner—sa
 - Database connect string—"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
 - Table name—stores
 - Trigger type—INSERT,DELETE
3. Commit the job.

Example: Monitor a Microsoft SQL Server Database Table for Specific Changes

Suppose that you want a job to monitor the sales table for changes to the ord_date and qty columns. The job runs under the sa user and completes only when both columns have changed.

To monitor a Microsoft SQL Server database table for specific changes

1. Create a Database Trigger job.
2. Enter the following properties:
 - Name—changes
 - Send to machine—DB_AGENT
 - Owner—sa
 - Database connect string—"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
 - Table name—sales
 - Trigger type—UPDATE
 - Trigger condition—"UPDATE(ord_date) and UPDATE(qty)"
3. Commit the job.

Example: Monitor a Microsoft SQL Server Database Table for Added Rows with a Trigger Condition

Suppose that you want a job to monitor the sales table for added rows. When the quantity for inserted title ID TC7777 is greater than or equal to 20, the job completes.

To monitor a Microsoft SQL Server database table for added rows with a trigger condition

1. Create a Database Trigger job.
2. Enter the following properties:
 - Name—insert
 - Send to machine—DB_AGENT
 - Owner—sa
 - Database connect string—"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
 - Table name—sales
 - Trigger type—INSERT
 - Trigger condition—'(select QTY from INSERTED where TITLE_ID='TC7777')>=20'
3. Commit the job.

Database Trigger Job Examples for IBM DB2

The following examples describe sample Database Trigger jobs that monitor IBM DB2 database tables.

Example: Monitor an IBM DB2 Database Table for Added Rows with a Trigger Condition

Suppose that you want a job to monitor the STAFF table for added rows. When the total number of rows is greater than or equal to 37, the job completes.

To monitor an IBM DB2 database table for added rows with a trigger condition

1. Create a Database Trigger job.
2. Enter the following properties:
 - Name—addrw
 - Send to machine—DB_AGENT
 - Owner—entadm
 - Database connect string—"jdbc:db2://10.1.4.131:50000/SAMPLE"
 - Table name—staff
 - Trigger type—INSERT
 - Trigger condition—'(select count(*) from STAFF)>=37'
3. Commit the job.

Example: Monitor an IBM DB2 Database Table for Specific Changes

Suppose that you want to monitor the STAFF table for changes to the DEPT and JOB columns. The job runs on the DB_AGENT agent computer and connects to the SAMPLE database. The job runs under the entadm user and completes once DEPT or JOB is updated.

To monitor an IBM DB2 database table for specific changes

1. Create a Database Trigger job.
2. Enter the following properties:
 - Name—deleted
 - Send to machine—DB_AGENT
 - Owner—entadm
 - Database connect string—"jdbc:db2://10.1.4.131:50000/SAMPLE"
 - Table name—staff
 - Trigger type—UPDATE
 - Trigger condition—"UPDATE of DEPT, JOB"
3. Commit the job.

Database Trigger Properties

The Database Trigger category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Continuous

(Optional) Indicates whether to monitor for the conditions continuously if selected. Each time the specified conditions occur, an alert is written to the scheduler log file.

Default: unselected (the job does not monitor for the conditions)

Note: To end continuous monitoring, you must complete the job manually.

JIL attribute: continuous

Database connect string

(Optional) Specifies a JDBC database resource location (URL) for a job. Use the appropriate format for your database type, as follows:

- For an Oracle database:

"jdbc:oracle:thin:@host:port:dbname"

- For a Microsoft SQL Server database:

"jdbc:sqlserver://host:port;DatabaseName=dbname"

- For an IBM DB2 database:

"jdbc:db2://host:port/dbname"

Limits: Up to 256 characters; case-sensitive

JIL attribute: connect_string

Database type

Specifies whether the database is an Oracle, MSSQL, or DB2 database.

JIL attribute: dbtype

Oracle user role

(Optional) Specifies the role of the Oracle user to log in as. The user role must be defined in the Oracle database. The user role uses the following syntax:

as *userrole*

Limits: Up to 64 characters; case-sensitive

Example: as sysdba

JIL attribute: user_role

Table name

Specifies the name of the database table to monitor for changes.

Limits: Up to 300 characters

JIL attribute: tablename

Trigger condition

(Optional) Specifies the condition to monitor in the database, as follows:

- For Oracle and DB2, this condition is the WHEN clause.
- For SQL Server, this condition is the IF clause.

Limits: Up to 500 characters; case-sensitive

Note: For the specific database syntax, refer to your database vendor's documentation.

JIL attribute: trigger_cond

Trigger type

Specifies the type of database change to monitor for, as follows:

DELETE

Monitors for the deletion of a row in the database table.

INSERT

Monitors for the insertion of a row in the database table.

UPDATE

Monitors for an update to any of the rows in the database table.

Default: INSERT

Example: Selecting INSERT and UPDATE monitors for the insertion of a row or an update to any of the rows.

Note: You can select multiple trigger types.

JIL attribute: trigger_type

Define an SQL Job

You can define an SQL job to run an SQL query against an Oracle, Microsoft SQL Server, or IBM 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. 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. Create an SQL job in either Quick Edit or Application Editor.
The Properties section for the SQL job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

SQL command

Specifies an SQL command to run against database tables.

3. (Optional) Specify optional [SQL properties](#) (see page 238).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The SQL job is defined.

More information:

[SQL Job Examples for Oracle](#) (see page 232)

[SQL Job Examples for Microsoft SQL Server](#) (see page 234)

[SQL Job Examples for IBM DB2](#) (see page 236)

SQL Job Examples for Oracle

The following examples describe sample SQL jobs that run SQL commands against database tables.

Example: Add a Row to a Database Table

Suppose that you want a job to add a row of data to the emp table in an Oracle database.

To add a row to an Oracle database table

1. Create an SQL job.
2. Enter the following properties:
 - Name—addrow
 - Send to machine—DB_AGENT
 - SQL command—'INSERT INTO EMP(EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO) VALUES(2476, "robert", "sales", 435, "01-OCT-2005", 65000, 10, 75)'
 - Database connect string—"jdbc:oracle:thin:@myhost:1521:orcl"
 - Oracle user role—as sysdba
3. Commit the job.

Example: Update a Row in a Database Table

Suppose that you want a job to update a record in the emp table and change the salary to 75,000 for the employee with name robert. The table is stored in a Microsoft SQL Server database.

To update a row in a database table

1. Create an SQL job.
2. Enter the following properties:
 - Name—updaterow
 - Send to machine—DB_AGENT
 - SQL command—'UPDATE EMP SET SAL=75000 where ENAME="robert"'
 - Database connect string—"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
 - Oracle user role—scott
3. Commit the job.

Example: Delete a Row from a Database Table

Suppose that you want a job to delete a row from the emp table for the employee with name robert. The table is stored in an IBM DB2 database.

To delete a row from a database table

1. Create an SQL job.
2. Enter the following properties:
 - Name—deleterow
 - Send to machine—DB_AGENT
 - SQL command—'DELETE FROM EMP WHERE ENAME="robert"'
 - Database connect string—"jdbc:db2://10.1.4.131:50000/SAMPLE"
 - Oracle user role—scott
3. Commit the job.

Example: Return Data from an Oracle Database Table that Match a Condition

Suppose that you want a job to query the emp table for names that have salaries greater than 40,000. If the query returns a name that begins with the letter d, the job completes.

To return data from an Oracle database table that match a condition

1. Create an SQL job.
2. Enter the following properties:
 - Name—query
 - Send to machine—DB_AGENT
 - SQL command—'SELECT ENAME FROM EMP WHERE SAL > 40000'
 - Destination file—/emp/salary.txt
 - Success criteria—ENAME=d.*
 - Database connect string—"jdbc:oracle:thin:@myhost:1521:orcl"
 - Oracle user role—as sysdba
3. Commit the job.

For example, the salary.txt file contains the following output:

Output for: SELECT ENAME FROM EMP WHERE SAL > 40000

```
ENAME
-----
donald
```

SQL Job Examples for Microsoft SQL Server

The following examples describe sample SQL jobs that run SQL commands against Microsoft SQL Server database tables.

Example: Add a Row to a Microsoft SQL Server Database Table

Suppose that you want a job to add a row for a new store to the stores table.

To add a row to a Microsoft SQL Server database table

1. Create an SQL job.
2. Enter the following properties:
 - Name—addrow
 - Send to machine—DB_AGENT
 - Owner—sa
 - SQL command—'INSERT INTO stores(stor_id, stor_name, stor_address, city, state, zip) VALUES("6523", "Chapters", "6523 Main St.", "San Diego", "CA", "93223")'
 - Database connect string—"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
3. Commit the job.

Example: Update a Row in a Microsoft SQL Server Database Table

Suppose that you want a job to update the row in the sales table that matches order number 6871 and change the values for the date and quantity of that order.

To update a row in a Microsoft SQL Server database table

1. Create an SQL job.
2. Enter the following properties:
 - Name—updaterow
 - Send to machine—DB_AGENT
 - Owner—sa
 - SQL command—'UPDATE sales SET ord_date="6/15/2006", qty=10 WHERE ord_num="6871"'
 - Database connect string—"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
3. Commit the job.

Example: Delete a Row from a Microsoft SQL Server Database Table

Suppose that you want a job to delete the row for store ID 6523 from the stores table.

To delete a row from a Microsoft SQL Server database table

1. Create an SQL job.
2. Enter the following properties:
 - Name—delrow
 - Send to machine—DB_AGENT
 - Owner—sa
 - SQL command—'DELETE FROM stores WHERE stor_id="6523"'
 - Database connect string—"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
3. Commit the job.

Example: Return Data from a Microsoft SQL Server Database Table that Matches a Condition

Suppose that you want a job to query the sales table for orders that have a quantity greater than 20. The order numbers that match the query appear in the output file ordnum.txt.

To return data from a Microsoft SQL Server database table that matches a condition

1. Create an SQL job.
2. Enter the following properties:
 - Name—delrow
 - Send to machine—DB_AGENT
 - Owner—sa
 - SQL command—'SELECT ORD_NUM FROM SALES WHERE QTY > 20'
 - Destination file—C:\sales\ordnum.txt
 - Success criteria—ORD_NUM=A2976
 - Database connect string—"jdbc:sqlserver://myhost:1433;DatabaseName=pubs"
3. Commit the job.

The ordnum.txt file contains the following order numbers:

```
A2976  
QA7442.3  
P2121  
N914014  
P3087a  
P3087a  
X999  
P723  
QA879.1
```

Because the query returns an order number that matches the job criteria A2976, the job completes.

Suppose we change the job criteria statement in the above example to the following:

```
JOB_CRITERIA .*B[0-9]
```

In this case, the query would still return the same order numbers, but the job would fail because it would not find a matching order number containing the letter B and followed by a number.

SQL Job Examples for IBM DB2

The following examples describe sample SQL jobs that run SQL commands against IBM DB2 database tables.

Example: Add a Row to an IBM DB2 Database Table

Suppose that you want a job to add a row of data to the STAFF table under the user entadm.

To add a row to an IBM DB2 database table

1. Create an SQL job.
2. Enter the following properties:
 - Name—addrow
 - Send to machine—DB_AGENT
 - Owner—entadm
 - SQL command—'INSERT into ENTADM.STAFF(ID, NAME, DEPT, JOB, YEARS, SALARY, COMM) VALUES(556, "Jonson", 84, "Sales", 1, 40500.50, 100)'
 - Database connect string—"jdbc:db2://10.1.4.131:50000/SAMPLE"
3. Commit the job.

Example: Update a Row in an IBM DB2 Database Table

Suppose that you want a job to update a record in the STAFF table under the user entadm. The job changes the years to 3 for the employee with the name Jonson.

To update a row in an IBM DB2 database table

1. Create an SQL job.
2. Enter the following properties:
 - Name—update
 - Send to machine—DB_AGENT
 - Owner—entadm
 - SQL command—'UPDATE ENTADM.STAFF SET YEARS=3 where NAME="Jonson"'
 - Database connect string—"jdbc:db2://10.1.4.131:50000/SAMPLE"
3. Commit the job.

Example: Delete a Row from an IBM DB2 Database Table

Suppose that you want a job to delete a row from the STAFF table under the user entadm for the employee with the name Jonson.

To delete a row from an IBM DB2 database table

1. Create an SQL job.
2. Enter the following properties:
 - Name—delrow
 - Send to machine—DB_AGENT
 - Owner—entadm
 - SQL command—'DELETE FROM ENTADM.STAFF where NAME="Jonson"'
 - Database connect string—"jdbc:db2://10.1.4.131:50000/SAMPLE"
3. Commit the job.

Example: Return Data from an IBM DB2 Database Table that Matches a Condition

Suppose that you want a job to query the STAFF table under the user entadm for employees that have salaries greater than 40,000. If the query returns an employee name that begins with the letter J, the job completes.

To return data from an IBM DB2 database table that matches a condition

1. Create an SQL job.
2. Enter the following properties:
 - Name—data
 - Send to machine—DB_AGENT
 - Owner—entadm
 - SQL command—'SELECT NAME FROM ENTADM.STAFF where SALARY > 40000'
 - Destination file—/staff/salary.txt
 - Success criteria—NAME=J.*
 - Database connect string—"jdbc:db2://10.1.4.131:50000/SAMPLE"
3. Commit the job.

For example, the salary.txt file contains the following output:

Output for: SELECT NAME FROM ENTADM.STAFF where SALARY > 40000

| NAME |
|--------|
| ----- |
| Jonson |

SQL Properties

The SQL category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Database connect string

(Optional) Specifies a JDBC database resource location (URL) for a job. Use the appropriate format for your database type, as follows:

- For an Oracle database:

"jdbc:oracle:thin:@host:port:dbname"

- For a Microsoft SQL Server database:

"jdbc:sqlserver://host:port;DatabaseName=dbname"

- For an IBM DB2 database:

"jdbc:db2://host:port/dbname"

Limits: Up to 256 characters; case-sensitive

JIL attribute: connect_string

Destination file

(Optional) Specifies the output destination file that stores the SQL query results.

Limits: Up to 256 characters; case-sensitive

JIL attribute: destination_file

SQL command

Specifies an SQL command to run against database tables.

Limits: Up to 1000 characters

Note: The value can contain any SQL command including SELECT, DELETE, UPDATE, or INSERT.

JIL attribute: sql_command

Success criteria

(Optional) Defines a regular expression that is used to evaluate a return string. If the return string matches the regular expression, the job completes successfully. Otherwise, the job fails.

Limits: Up to 500 characters; case-sensitive

Note: Each return string includes the field name from the SELECT statement and its value, separated by an equal sign (=). For example, consider the query SELECT ORD_NUM FROM SALES. To match order number A2976, specify the regular expression ORD_NUM=A2976. Specifying the regular expression A2976 does not match any return string causing the job to fail.

JIL attribute: success_criteria

Oracle user role

(Optional) Specifies the role of the Oracle user to log in as. The user role must be defined in the Oracle database. The user role uses the following syntax:

as *userrole*

Limits: Up to 64 characters; case-sensitive

Example: as sysdba

JIL attribute: user_role

Chapter 14: Monitoring Jobs

This section contains the following topics:

- [Monitoring Jobs](#) (see page 241)
- [File Trigger Jobs](#) (see page 242)
- [Windows Event Log Monitoring Jobs](#) (see page 243)
- [Define a CPU Monitoring Job](#) (see page 244)
- [Define a Disk Monitoring Job](#) (see page 252)
- [Define a File Trigger Job](#) (see page 261)
- [Define an IP Monitoring Job](#) (see page 271)
- [Define a Process Monitoring Job](#) (see page 274)
- [Define a Text File Reading and Monitoring Job](#) (see page 279)
- [Define a Windows Event Log Monitoring Job](#) (see page 286)
- [Define a Windows Service Monitoring Job](#) (see page 292)

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.

File Trigger Jobs

File Trigger jobs let you monitor file activity. You can define File Trigger jobs for UNIX, Windows, or i5/OS systems.

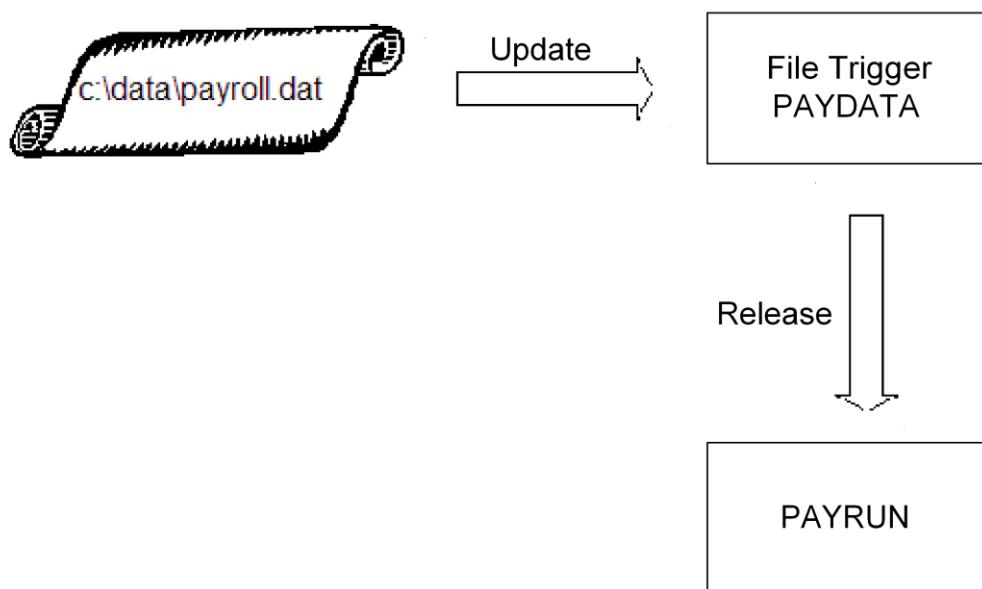
Note: To run these jobs, your system requires CA WA Agent for UNIX, Linux, Windows, or i5/OS.

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:



Windows Event Log Monitoring Jobs

Windows Event Log Monitoring (OMEL) jobs monitor Windows event logs on a local computer. This monitor returns the first event available in a particular Windows event log.

Windows 2000 computers and later versions keep at least three kinds of event logs, including the following:

- Application log—Contains an event, application or program log. For example, a database program might record a file error in the application log.
- System log—Contains events the Windows 2000 system components log. For example, the failure of a driver or other system component to load during startup is recorded in the system log.
- Security log—Records security events (such as valid and invalid logon attempts) and events related to resource use (such as creating, opening or deleting files).

The Windows Event Log Monitoring job only monitors event logs maintained by the operating system and available in the Event Viewer. For more information about the Windows event properties stored in Windows event logs, view the Windows event logs on your local machine. To do so, select Start, Settings, Control Panel, Administrative Tools, Event Viewer. Then, select one of the three event categories and double-click any event to view a property page.

Note: To run these jobs, your system requires CA WA Agent for Windows.

Define a CPU Monitoring Job

You can define a CPU Monitoring job to monitor the CPU usage of the computer the specified agent is installed on.

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. Create a CPU Monitoring (OMCPU) job in either Quick Edit or Application Editor.
The Properties section for the CPU Monitoring job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

3. (Optional) Specify optional CPU Monitoring properties.

The CPU Monitoring category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

CPU Usage

(Defaulted) Specifies whether the job monitors the available or used CPU processing capacity. The following values are available:

FREE

Specifies that the job monitors the available CPU processing capacity.

USED

Specifies that the job monitors the used CPU processing capacity.

Default: FREE

Note: To use this property, you must specify WAIT or CONTINUOUS for the Monitor mode.

JIL attribute: cpu_usage

Inside range

(Optional) Indicates whether the job triggers when the value of CPU usage is within the range specified by the lower and upper boundary properties.

Default: selected (inside the range)

Note: To use this property, you must specify WAIT or CONTINUOUS for the Monitor mode.

JIL attribute: inside_range

Lower boundary (%)

(Optional) Defines the lower boundary of CPU usage in percent.

Limits: 1-100

Default: 0

Notes:

- If you specify a lower boundary without an upper boundary, the monitoring takes place between the lower boundary and 100 percent.
- An inside range is used for the boundary unless you specify outside for the range.
- To use this property, you must specify WAIT or CONTINUOUS for the Monitor mode.

JIL attribute: lower_boundary

Monitor mode

(Defaulted) Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately. The following values are available:

CONTINUOUS

Monitors for the conditions continuously. Each time the specified conditions occur, an alert is written to the log file.

Notes:

- When using the CONTINUOUS value in CPU Monitoring jobs, you must also specify the lower boundary, the upper boundary, or both.
- To end continuous monitoring, you must complete the job manually.

NOW

Checks for the conditions immediately. If the conditions are met, the job completes successfully. If the conditions are not met, the job fails.

WAIT

Waits for the conditions to occur. When the conditions are met, the job completes.

Default: WAIT

JIL attribute: monitor_mode

No change (%)

(Optional) Specifies the percentage change in CPU usage that must occur to trigger the job.

JIL attribute: no_change

Poll interval (secs)

(Optional) Specifies the interval (in seconds) between successive scans for a job that uses the CONTINUOUS monitor mode.

Note: The default is set on the agent by the objmon.scaninterval parameter.

JIL attribute: poll_interval

Upper boundary (%)

(Optional) Defines the upper boundary of CPU usage in percent.

Limits: 1-100

Default: 100

Notes:

- To use this property, you must specify WAIT or CONTINUOUS for the Monitor mode.
- If you specify an upper boundary without a lower boundary, the monitoring takes place between zero percent and the upper boundary.
- An inside range is used for the boundary unless you specify outside for the range.

JIL attribute: upper_boundary

1. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
2. Commit the job.

The CPU Monitoring job is defined.

More information:

[CPU Monitoring Job Examples](#) (see page 248)

CPU Monitoring Job Examples

The following example describes a sample CPU Monitoring job.

Example: Monitor CPU Availability Continuously

In this example, the agent returns information on the CPU capacity that is used within the range specified by the lower and upper boundaries.

To monitor CPU availability continuously

1. Create a CPU Monitoring job.
2. Enter the following properties:
 - Name—unix_textfile
 - Send to machine—AGENT
 - CPU Usage—USED
 - Monitor mode—CONTINUOUS
 - Lower boundary (%)—75
 - Upper boundary (%)—100
3. Commit the job.

CPU Monitoring Properties

The CPU Monitoring category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

CPU Usage

(Defaulted) Specifies whether the job monitors the available or used CPU processing capacity. The following values are available:

FREE

Specifies that the job monitors the available CPU processing capacity.

USED

Specifies that the job monitors the used CPU processing capacity.

Default: FREE

Note: To use this property, you must specify WAIT or CONTINUOUS for the Monitor mode.

JIL attribute: cpu_usage

Inside range

(Optional) Indicates whether the job triggers when the value of CPU usage is within the range specified by the lower and upper boundary properties.

Default: selected (inside the range)

Note: To use this property, you must specify WAIT or CONTINUOUS for the Monitor mode.

JIL attribute: inside_range

Lower boundary (%)

(Optional) Defines the lower boundary of CPU usage in percent.

Limits: 1-100

Default: 0

Notes:

- If you specify a lower boundary without an upper boundary, the monitoring takes place between the lower boundary and 100 percent.
- An inside range is used for the boundary unless you specify outside for the range.
- To use this property, you must specify WAIT or CONTINUOUS for the Monitor mode.

JIL attribute: lower_boundary

Monitor mode

(Defaulted) Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately. The following values are available:

CONTINUOUS

Monitors for the conditions continuously. Each time the specified conditions occur, an alert is written to the log file.

Notes:

- When using the CONTINUOUS value in CPU Monitoring jobs, you must also specify the lower boundary, the upper boundary, or both.
- To end continuous monitoring, you must complete the job manually.

NOW

Checks for the conditions immediately. If the conditions are met, the job completes successfully. If the conditions are not met, the job fails.

WAIT

Waits for the conditions to occur. When the conditions are met, the job completes.

Default: WAIT

JIL attribute: monitor_mode

No change (%)

(Optional) Specifies the percentage change in CPU usage that must occur to trigger the job.

JIL attribute: no_change

Poll interval (secs)

(Optional) Specifies the interval (in seconds) between successive scans for a job that uses the CONTINUOUS monitor mode.

Note: The default is set on the agent by the objmon.scaninterval parameter.

JIL attribute: poll_interval

Upper boundary (%)

(Optional) Defines the upper boundary of CPU usage in percent.

Limits: 1-100

Default: 100

Notes:

- To use this property, you must specify WAIT or CONTINUOUS for the Monitor mode.
- If you specify an upper boundary without a lower boundary, the monitoring takes place between zero percent and the upper boundary.
- An inside range is used for the boundary unless you specify outside for the range.

JIL attribute: upper_boundary

Define a Disk Monitoring Job

On Windows and UNIX systems, you can define a Disk Monitoring 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 an auxiliary storage pool (ASP).

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. Create a Disk Monitoring (OMD) job in either Quick Edit or Application Editor.
The Properties section for the Disk Monitoring job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

Disk drive

Specifies the path to the disk, logical partition, or auxiliary storage pool to be monitored.

Disk space

Specifies whether the job monitors the available or used disk space.

Monitor mode

Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

3. (Optional) Specify optional [Disk Monitoring properties](#) (see page 257).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Disk Monitoring job is defined.

More information:

[Disk Monitoring Job Examples](#) (see page 253)

Disk Monitoring Job Examples

The following examples describe sample Disk Monitoring jobs.

Example: Monitor Available Space on a Windows Drive

In this example, the local C drive is monitored. The space available is monitored and the value is expressed in megabytes.

To monitor available space on a Windows drive

1. Create a Disk Monitoring job.
2. Enter the following properties:
 - Name—cdrive
 - Send to machine—AGENT
 - Disk drive—C
 - Disk format—MB
 - Disk space—FREE
 - Monitor mode—NOW
3. Commit the job.

Example: Monitor a Local Partition Continuously

In this example, a local partition is continuously monitored. The space used is monitored and the value is expressed as a percentage. When the value for disk space used falls between 90 and 100 percent, an alert (DISK) is issued.

To monitor a local partition continuously

1. Create a Disk Monitoring job.
2. Enter the following properties:
 - Name—partition
 - Send to machine—AGENT
 - Disk drive—/export/home
 - Disk format—PERCENT
 - Disk space—USED
 - Inside range—selected
 - Monitor mode—CONTINUOUS
 - Lower boundary—90
3. Commit the job.

Example: Monitor a Local Drive Continuously

In this example, the local C drive is continuously monitored. The space used is monitored and the value is expressed as a percentage. When the value of disk space used is less than 16 percent or greater than 95 percent, an alert (DISK) is issued.

To monitor a local drive continuously

1. Create a Disk Monitoring job.
2. Enter the following properties:
 - Name—local
 - Send to machine—AGENT
 - Disk drive—C
 - Disk format—PERCENT
 - Disk space—USED
 - Monitor mode—CONTINUOUS
 - Lower boundary—16
 - Upper boundary—95
3. Commit the job.

Example: Monitor the System Auxiliary Storage Pool (ASP)

In this example, the agent monitors the system ASP. As specified in the DISK statement, the space available is monitored and the value is expressed in megabytes. The upper and lower boundaries are not specified, so the job returns immediately with the current status of the space being monitored.

The system ASP is denoted by the forward slash (/). You can also specify the system ASP as /dev/QASP01.

To monitor the system ASP

1. Create a Disk Monitoring job.
2. Enter the following properties:
 - Name—ASP
 - Send to machine—AGENT
 - Disk drive—/
 - Disk format—MB
 - Disk space—FREE
 - Monitor mode—NOW
3. Commit the job.

Example: Monitor an ASP for a Storage Space Value within a Range

In this example, the agent monitors the auxiliary storage pool (ASP) named QASP02. The space used is monitored and the value is expressed as a percentage. When 90 to 100 percent of the storage space is used, the job completes.

To monitor an ASP for a storage space value within a range

1. Create a Disk Monitoring job.
2. Enter the following properties:
 - Name—AUXASP
 - Send to machine—AGENT
 - Disk drive—/dev/QASP02
 - Disk format—PERCENT
 - Disk space—USED
 - Inside range—selected
 - Monitor mode—NOW
 - Lower boundary—90
 - Upper boundary—100
3. Commit the job.

Example: Monitor a Mounted File System for a Storage Space Value Outside of a Range

In this example, the agent monitors a Network File System (NFS) server mounted at /u1. The space used is monitored and the value is expressed as a percentage. When the value of storage space used is less than 16 percent or greater than 95 percent, the job completes.

To monitor a mounted file system for a storage space value outside of a range

1. Create a Disk Monitoring job.
2. Enter the following properties:
 - Name—NFS
 - Send to machine—AGENT
 - Disk drive—/u1
 - Inside range—not selected
 - Disk format—PERCENT
 - Disk space—USED
 - Monitor mode—NOW
 - Lower boundary—16
 - Upper boundary—95
3. Commit the job.

Disk Monitoring Properties

The Disk Monitoring category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Disk drive

Specifies the path to the disk, logical partition, or auxiliary storage pool to be monitored.

Limits: Up to 256 characters; case-sensitive

Example: /dev/QASP02

UNIX/Windows: Specify the path to the disk or logical partition to monitor.

i5/OS:

- Specify the path to a file system mounted on the i5/OS operating system.
- To specify the system ASP, type the forward slash (/).
- To specify any other ASP, use the following format, where *digit1* and *digit2* indicate the ASP number:

/dev/QASP*digit1**digit2*

JIL attribute: disk_drive

Disk format

(Defaulted) Specifies the unit of measurement used to monitor available or used disk space. The following values are available:

B—Monitors disk usage in bytes.

GB—Monitors disk usage in gigabytes.

KB—Monitors disk usage in kilobytes.

MB—Monitors disk usage in megabytes.

PERCENT—Monitors disk usage by percentage.

Default: PERCENT

JIL attribute: disk_format

Disk space

(Defaulted) Specifies whether the job monitors the available or used disk space. The following values are available:

FREE

Specifies that the job reads the available disk space for monitoring.

USED

Specifies that the job reads the used disk space for monitoring.

Default: FREE

JIL attribute: disk_space

Inside range

(Optional) Indicates whether the job completes when the disk space value is within the range specified by the lower and upper boundary properties.

Default: selected (inside the range)

JIL attribute: inside_range

Lower boundary

(Optional) Defines the start of the range to be searched. The Disk format property specifies the units for this value.

Limits: Up to 20 characters

Note: This value is required when the monitor mode is set to WAIT or CONTINUOUS.

JIL attribute: lower_boundary

Monitor mode

(Defaulted) Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately. The following values are available:

CONTINUOUS

Monitors for the conditions continuously. Each time the specified conditions occur, an alert is written to the log file.

Note: To end continuous monitoring, you must complete the job manually.

NOW

Checks for the conditions immediately. If the conditions are met, the job completes successfully. If the conditions are not met, the job fails.

WAIT

Waits for the conditions to occur. When the conditions are met, the job completes.

Note: When using the WAIT value, you can specify a lower boundary but you cannot specify an upper boundary.

Default: WAIT

JIL attribute: monitor_mode

No change (%)

(Optional) Specifies the percentage value that the disk usage must change to complete the job. If the change in disk usage is within this specified value, the job does not complete.

JIL attribute: no_change

Poll interval (secs)

(Optional) Specifies the interval (in seconds) between successive scans when the monitor mode is CONTINUOUS.

Note: This property can be set on the agent using the objmon.scaninterval parameter. The default value set on the agent is 10 seconds.

JIL attribute: poll_interval

Upper boundary

(Optional) Defines the end of the range to be searched. The Disk format property specifies the units for this value.

Limits: Up to 20 characters

Note: This value is required when the monitor mode is set to WAIT but must be blank when set to CONTINUOUS.

JIL attribute: upper_boundary

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. Create a File Trigger (FT) job in either Quick Edit or Application Editor.
The Properties section for the File Trigger job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

File(s) to watch

Specifies the path to and name of the file to monitor.

3. (Optional) Specify optional [File Trigger properties](#) (see page 266).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The File Trigger job is defined.

More information:

[File Trigger Properties](#) (see page 266)

File Trigger Job Examples

The following examples describe sample File Trigger jobs that monitor file activity.

Example: Monitor Daily Update to Payroll File on a Windows Computer

Suppose that you want a job to monitor a daily update to a payroll.dat file on a Windows computer. When the file is updated, the job completes.

To monitor daily update to payroll file on a Windows computer

1. Create a File Trigger job.
2. Enter the following properties:
 - Name—PAYDATA
 - Send to machine—NT_NY
 - Application—PAYROLL
 - File(s) to watch—c:\data\payroll.dat
 - Trigger type—Update
3. Commit the job.

Example: Monitor Daily Update to Payroll File on a UNIX Computer

Suppose that you want a job to monitor a daily update to a payroll file on a UNIX computer. When the file is updated, the job completes.

To monitor daily update to payroll file on a UNIX computer

1. Create a File Trigger job.
2. Enter the following properties:
 - Name—PAYDATA
 - Send to machine—UNIX_NY
 - Application—PAYROLL
 - File(s) to watch—/usr/data/payroll
 - Trigger type—Update
3. Commit the job.

Example: Monitor for the Creation of a File that Is Owned by a Specified UNIX User ID

Suppose that you want a job to monitor for the creation of the /data/payroll.dat file that is owned by the UNIX user ID JDOE:

- If the file does not exist when the job is readied, 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 USER criteria. If the owner of the file changes to JDOE, the job completes.

To monitor for the creation of a file that is owned by a specified UNIX user ID

1. Create a File Trigger job.
2. Enter the following properties:
 - Name—PAYDATA
 - Send to machine—SYSAGENT
 - Application—PAYROLL
 - File(s) to watch—/data/payroll.dat
 - File owner—JDOE
 - Trigger type—Create
3. Commit the job.

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.

To monitor for the existence of a file that is owned by a specified UNIX group

1. Create a File Trigger job.
2. Enter the following properties:
 - Name—PAYDATA
 - Send to machine—SYSAGENT
 - Application—PAYROLL
 - File(s) to watch—/data/payroll.dat
 - File group—ACCTS
 - Trigger type—Exist
3. Commit the job.

Example: Monitor a Daily Update to a Payroll File Object on an i5/OS System

Suppose that you want a job to monitor a daily update to a payroll file object on an i5/OS computer. When the file is updated, the job completes.

To monitor a daily update to a payroll file object on an i5/OS system

1. Create a File Trigger job.
2. Enter the following properties:
 - Name—JOBDATA
 - Send to machine—I5AGENT
 - Application—PAYROLL
 - File(s) to watch—/QSYS.LIB/LIBRARY.LIB/DEPT.FILE/PAYROLL.MBR
 - Trigger type—Update
3. Commit the job.

Example: Use Wildcards to Specify a File Name on the Root File System

In this example, the job monitors for files that are created in the /home/cybesp/ directory in the root file system. The job completes if a file is created with a file name that matches the following criteria:

- Starts with PID
- Ends with four characters
- Has any extension

To use wildcards to specify a file name on the root file system

1. Create a File Trigger job.
2. Enter the following properties:
 - Name—JOBDATA
 - Send to machine—I5AGENT
 - Application—NEWFILE
 - File(s) to watch—'/home/cybesp/PID????.*'
 - Trigger type—Create
3. Commit the job.

Example: Use a Generic Name to Specify a QSYS File Object

In this example, the job completes when any file object with a file name that starts with PAY and ends with any characters is created in the QSYS file system. Note that the file name is enclosed in single quotation marks so that the text following /* is not interpreted as a comment.

To use a generic name to specify a QSYS file object

1. Create a File Trigger job.
2. Enter the following properties:
 - Name—JOBDATA
 - Send to machine—I5AGENT
 - Application—NEWFILE
 - File(s) to watch—'LIB/PAY/*FILE'
 - Trigger type—Create
3. Commit the job.

Example: Monitor the Size of a QSYS File Object

In this example, the job completes when the EXPOBJ file object in the QSYS file system increases by 10 bytes or more. Since no member is specified in the file name, the job monitors the entire object size. The entire object size includes the size of the file object itself and the total size of the file object's members.

Notes:

- The file name is enclosed in single quotation marks so that the text following /* is not interpreted as a comment.
- To monitor only the total size of the file object's members, specify *ALL as the member name.

To monitor the size of a QSYS file object

1. Create a File Trigger job.
2. Enter the following properties:
 - Name—JOBDATA
 - Send to machine—I5AGENT
 - Application—NEWFILE
 - File(s) to watch—'LIB/EXPOBJ/*FILE'
 - Trigger type—Expand
 - Change type—Size
 - Change value—10
3. Commit the job.

File Trigger Properties

The File Trigger category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Change type

(Optional) Specifies the type of change to detect when monitoring a file for an increase or decrease in size.

The following options are available:

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.

Note: Some file systems will create files with extra space in anticipation of further file expansion requests. Therefore, specifying exact byte counts will not always work.

Default: Size

Notes:

- This field is required when the type of file activity is Expand or Shrink.
- The change type works in concert with the change value; if one is specified, the other must be also.

JIL attribute: watch_file_change_type

Change value

(Optional) Defines one of the following:

- The change in file size when the file change type is Delta or Percent.
- The limit of the file size when the file change type is Size. If the type of file activity is Expand, the file trigger occurs when the file size is equal to or is greater than the specified size. If the type of file activity is Shrink, the file trigger occurs when the file size is equal to or less than the specified size.

Notes:

- This field is required when the type of file activity is Expand or Shrink.
- This value corresponds to the type of file activity.

JIL attribute: watch_file_change_value

Continuous

(Optional) Indicates whether the job should wait for the condition to be satisfied before it ends.

Default: unselected (the job does not wait)

Note: To end continuous monitoring, you must complete the job manually.

JIL attribute: continuous

File(s) to watch

Specifies the path to and name of the files to monitor. To specify multiple files, use the asterisk (*) and question mark (?) wildcard characters. Use * for any number of characters and ? for any single character. Do not use the * and ? in the path.

Limits: Up to 255 characters; case-sensitive

UNIX Notes:

- You can specify variables exported from the profile script or global variables in the name.
- Wildcard characters are expanded according to the wildcard expansion rules of the Bourne Shell.

Windows Notes:

- You can specify system environment variables, job profile environment variables, and global variables in the name. If the variable is referenced in the job profile, we recommend that you enclose the variable in braces (for example, \${PATH}). Use the expression \$\$global_name for global variables.
- When specifying drive letters in job definitions, you must escape the colon with quotation marks or backslashes. For example, C:\tmp or "C:\tmp" is valid; C:\tmp is not.

Example: /tmp/a*.data

JIL attribute: watch_file

File group

(Optional) Specifies the name of the group that owns the file to be monitored.

Limits: Up to 80 characters

JIL attribute: watch_file_groupname

File owner

(Optional) Specifies the user ID that owns the file to be monitored.

Limits: Up to 80 characters

JIL attribute: watch_file_owner

Trigger type

(Optional) Specifies the type of file activity that a File Trigger job monitors for.

The following options are available:

Create

Indicates that the file trigger occurs when the file is created.

Delete

Indicates that the file trigger occurs when the file is deleted.

Exist

Completes the job if the file exists when the file trigger is set.

Expand

Indicates that the file trigger occurs when the file size increases.

Notexist

Completes the job if the file does not exist when the file trigger is set.

Shrink

Indicates that the file trigger occurs when the file size decreases.

Update

Indicates that the file trigger occurs when the file is updated.

Default: CREATE

JIL attribute: watch_file_type

Watch no change (min)

(Optional) Specifies the interval (in minutes) the File Trigger job should check for the existence and size of the watched file.

Default: 60

Note: This field is required when the type of file activity is Create or Update.

JIL attribute: watch_no_change

Watch recursive

(Optional) Indicates whether a File Trigger job monitors for file activity in the specified directory only or in the specified directory and all of its subdirectories.

Default: unselected (the job does not monitor for file activity in the specified directory only)

JIL attribute: watch_file_recursive

Windows user

(Optional) Specifies the user ID that is used for monitoring UNC files (Windows only).

Limits: Up to 80 characters

JIL attribute: watch_file_win_user

Define an IP Monitoring Job

You can define an IP Monitoring job to monitor an IP address or a port on 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. Create an IP Monitoring (OMIP) job in either Quick Edit or Application Editor.
The Properties section for the IP Monitoring job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

IP host

Specifies the DNS name or IP address.

3. (Optional) Specify optional [IP Monitoring properties](#) (see page 272).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The IP Monitoring job is defined.

More information:

[IP Monitoring Job Examples](#) (see page 272)

IP Monitoring Job Examples

The following example describes a sample IP Monitoring job.

Example: Monitor an Agent IP Address

You can use an IP Monitoring job to find out if an agent is running. In this example, the agent IP address is 172.24.2.20 and the input port is 9401. The following job completes if the agent is running.

To monitor an agent IP address

1. Create an IP Monitoring job.
2. Enter the following properties:
 - Name—agent_running
 - Send to machine—AGENT
 - IP host—172.24.2.20
 - IP port—9401
 - IP status—RUNNING
3. Commit the job.

IP Monitoring Properties

The IP Monitoring category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

IP host

Specifies the DNS name or IP address.

Limits: Up to 100 characters; case-sensitive

JIL attribute: ip_host

IP port

(Optional) Specifies the port number for the IP address being monitored. The agent attempts to connect to this port.

Limits: 0-65535

Note: If you specify 0 or no value for the port, the agent uses ping for monitoring.

JIL attribute: ip_port

IP status

(Defaulted) Specifies the status of the IP address to monitor, as follows:

RUNNING

Specifies that the job monitors the IP address for a running status.

STOPPED

Specifies that the job monitors the IP address for a stopped status.

Default: STOPPED

JIL attribute: ip_status

Monitor mode

(Defaulted) Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

NOW

Checks for the conditions immediately. If the conditions are met, the job completes successfully. If the conditions are not met, the job fails.

WAIT

Waits for the conditions to occur. When the conditions are met, the job completes.

Default: NOW

JIL attribute: monitor_mode

Define a Process Monitoring Job

You can define a Process Monitoring job to monitor process execution 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. Create a Process Monitoring (OMP) job in either Quick Edit or Application Editor.

The Properties section for the Process Monitoring job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Process name

Specifies the name of the process to be monitored.

3. (Optional) Specify optional [Process Monitoring properties](#) (see page 277).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Process Monitoring job is defined.

More information:

[Process Monitoring Job Examples](#) (see page 275)

Process Monitoring Job Examples

The following examples describe sample Process Monitoring jobs.

Example: Monitor the cybAgent.exe Process on Windows

In this example, the agent monitors the cybAgent.exe process. If the cybAgent.exe process is running, the job completes successfully.

To monitor the cybAgent.exe process on Windows

1. Create a Process Monitoring job.
2. Enter the following properties:
 - Name—cybAgent_Win
 - Send to machine—AGENT
 - Process name—'c:\Program Files\CA\WA System Agent\cybAgent.exe'
 - Process status—RUNNING
3. Commit the job.

Example: Monitor the cybAgent Process on UNIX

In this example, the agent monitors the cybAgent process. If the cybAgent process is running, the job completes successfully.

Note: Alternatively, you can also specify a PID number in place of the process name.

To monitor the cybAgent process on UNIX

1. Create a Process Monitoring job.
2. Enter the following properties:
 - Name—cybAgent_UNIX
 - Send to machine—AGENT
 - Process name—cybAgent
 - Process status—RUNNING
3. Commit the job.

Example: Monitor a Process that Has Any Job Number and Is Running Under Any User Name

In this example, the agent monitors for any process named CYBAGENT, regardless of the first part (job number) and the second part (user name) of the process ID. The job completes successfully if at least one process named CYBAGENT is running.

To monitor a process that has any job number and is running under any user name

1. Create a Process Monitoring job.
2. Enter the following properties:
 - Name—CYBAGENT
 - Send to machine—i5AGENT
 - Process name—'*ALL/*ALL/CYBAGENT'
 - Process status—RUNNING
3. Commit the job.

Example: Monitor Multiple Processes that Have Similar Names

In this example, the agent monitors all processes running under the JDOE user profile with names that start with CALC. When all of these processes stop running, the job completes successfully.

To monitor multiple processes that have similar names

1. Create a Process Monitoring job.
2. Enter the following properties:
 - Name—CALC
 - Send to machine—i5AGENT
 - Process name—'*ALL/JDOE/CALC*'
 - Process status—STOPPED
3. Commit the job.

Process Monitoring Properties

The Process Monitoring category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Monitor mode

(Defaulted) Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

NOW

Checks for the conditions immediately. If the conditions are met, the job completes successfully. If the conditions are not met, the job fails.

WAIT

Waits for the conditions to occur. When the conditions are met, the job completes.

Default: NOW

JIL attribute: monitor_mode

Process name

Specifies the name of the process to be monitored.

Limits: Up to 256 characters; case-sensitive

UNIX: Specify a PID or process name.

Windows: Specify a full path or process name.

JIL attribute: process_name

Process status

(Defaulted) Specifies the status of the process to be monitored, as follows:

RUNNING

Specifies that the job monitors the process for a running status.

STOPPED

Specifies that the job monitors the process for a stopped status.

Default: STOPPED

JIL attribute: process_status

Define a Text File Reading and Monitoring Job

You can define a Text File Reading and Monitoring job to search a text file on a Windows or a UNIX 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. Create a Text File Reading and Monitoring (OMTF) job in either Quick Edit or Application Editor.

The Properties section for the Text File Reading and Monitoring job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

Text file name

Specifies the path to and name of the text file to search.

Text file filter

Defines the text string to search for. You can specify the text string as a regular expression.

Monitor mode

Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

Text file mode

Specifies the search mode when monitoring a text file.

3. (Optional) Specify optional [Text File Reading and Monitoring properties](#) (see page 282).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Text File Reading and Monitoring job is defined.

More information:

[Text File Reading and Monitoring Job Examples](#) (see page 280)

Text File Reading and Monitoring Job Examples

The following examples describe sample Text File Reading and Monitoring jobs.

Example: Search for a Text String on a Windows System

In this example, the job searches the text file component.txt, between lines 1 and 20, for the text string "CreateService failed". When the text string is found, the job completes.

To search for a text string on a Windows system

1. Create a Text File Reading and Monitoring job.
2. Enter the following properties:
 - Name—component
 - Send to machine—AGENT
 - Text file name—"c:\Program Files\CA\component.txt"
 - Text file filter—"CreateService failed"
 - Text file filter exists—selected
 - Monitor mode—NOW
 - Text file mode—LINE
 - Lower boundary—1
 - Upper boundary—20
3. Commit the job.

Example: Search for a Text String on an i5/OS System

In this example, the job searches the DATA member, between lines 1 and 20, for the text string "Create file failed". When the text string is found, the job completes.

To search for a text string on an i5/OS system

1. Create a Text File Reading and Monitoring job.
2. Enter the following properties:
 - Name—DATA
 - Send to machine—I5AGENT
 - Text file name—/QSYS.LIB/LIBRARY.LIB/RESULTS.FILE/DATA.MBR
 - Text file filter—'Create file failed'
 - Text file filter exists—selected
 - Monitor mode—NOW
 - Text file mode—Line
 - Lower boundary—1
 - Upper boundary—20
3. Commit the job.

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 operand REGEX in the SEARCHRANGE statement.

To search for a regular expression

1. Create a Text File Reading and Monitoring job.
2. Enter the following properties:
 - Name—regex
 - Send to machine—AGENT
 - Text file name—'/home/cybesp/file.txt'
 - Text file filter—'^\w{4,10}\.'
 - Text file filter exists—selected
 - Monitor mode—NOW
 - Text file mode—REGEX
 - Lower boundary—(\A\W\sE)
3. Commit the job.

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 following syntax:

TEXTSTRING '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.

Text File Reading and Monitoring Properties

The Text File Reading and Monitoring category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Encoding

(Optional) Specifies the name of the character set used to encode the data in the file.

Default: US-ASCII

Limits: Up to 42 characters

JIL attribute: encoding

Lower boundary

(Optional) Defines the start of the range to be searched. The format of this value depends on the text file search mode. The formatting options are as follows:

- Numeric—Specify a numeric value if the search mode is LINE.
- Regular expression—Specify a regular expression if the search mode is REGEX.
- Date and time—Specify date and time values if the search mode is DATETIME. Define the date and time using the format specified by the time format.

Limits: Up to 256 characters; case-sensitive

JIL attribute: lower_boundary

Monitor mode

Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately. The options are as follows:

CONTINUOUS

Monitors for the conditions continuously. Each time the specified conditions occur, an alert is written to the log file.

Notes:

- When using the CONTINUOUS value, you can specify a lower boundary but you cannot specify an upper boundary.
- To end continuous monitoring, you must complete the job manually.

NOW

Checks for the conditions immediately. If the conditions are met, the job completes successfully. If the conditions are not met, the job fails.

WAIT

Waits for the conditions to occur. When the conditions are met, the job completes.

Note: When using the WAIT value, you can specify a lower boundary but you cannot specify an upper boundary.

Default: WAIT

JIL attribute: monitor_mode

Text file filter

Defines the text string to search for. You can specify the text string as a regular expression.

Limits: Up to 1024 characters; case-sensitive

JIL attribute: text_file_filter

Text file exists

(Optional) Indicates how the job monitors a specified text file for a text string specified in the Text file filter field.

- Selected—Monitors whether a text string exists in a specified text file. If the text string exists, the job completes successfully, or an alert is triggered (if monitoring continuously).
- Unselected—Monitors whether a text string does not exist in a specified text file. If the text string does not exist, the job completes successfully. If the text string exists, the job fails.

Default: selected (monitors for a text string)

JIL attribute: text_file_filter_exists

Text file mode

Specifies the search mode when monitoring a text file.

LINE

Searches for a text string between line numbers that define the upper and lower boundaries to search in the text file.

REGEX

Searches for a text string between regular expressions that define the lower and upper boundaries to search in the text file.

DATETIME

Searches for a text string between a date range specified by the upper and lower boundaries.

Default: LINE

JIL attribute: text_file_mode

Text file name

Specifies the path to and name of the text file to search.

Limits: Up to 256 characters; case-sensitive

JIL attribute: text_file_name

Time format

(Optional) Defines the date and time pattern to use when the upper and lower boundaries are specified as date and time. The upper and lower boundaries are used to search inside a log file.

Limits: Up to 256 characters; case-sensitive

Notes:

- To specify the time format, construct a time pattern string that is used as a mask for searching the particular time stamp you want to find. In this pattern, all ASCII letters are reserved as pattern letters.
- For a list of symbols you can use to build a time format pattern and examples, refer to the CA Workload Automation AE *Reference Guide* for the JIL attribute.

JIL attribute: time_format

Time position

(Optional) Specifies the first column of the time stamp in the log file.

Limits: 0-99999

JIL attribute: time_position

Upper boundary

(Optional) Defines the end of the range to be searched. The format of this value depends on the text file search mode. The formatting options are as follows:

- Numeric—Specify a numeric value if the search mode is LINE.
- Regular expression—Specify a regular expression if the search mode is REGEX.
- Date and time—Specify date and time values if the search mode is DATETIME. Define the date and time using the format specified by the time format.

Limits: Up to 256 characters; case-sensitive

Note: The upper boundary can only be set when the monitor mode is NOW.

JIL attribute: upper_boundary

Define a Windows Event Log Monitoring Job

You can define a Windows Event Log Monitoring job to monitor a Windows event log on a local computer. 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. Create a Windows Event Log Monitoring (OMEL) job in either Quick Edit or Application Editor.

The Properties section for the Windows Event Log Monitoring job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

Log name

Specifies the name of the event log.

3. (Optional) Specify optional [Windows Event Log Monitoring properties](#) (see page 288).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Windows Event Log Monitoring job is defined.

More information:

[Windows Event Log Monitoring Job Examples](#) (see page 287)

Windows Event Log Monitoring Job Examples

The following examples describe sample Windows Event Log Monitoring jobs.

Example: Monitor an Application Log Continuously

In this example, the event log for applications is monitored continuously and the agent issues an alert ELOG for all instances of the Information 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.

To monitor an application log continuously

1. Create a Windows Event Log Monitoring job.
2. Enter the following properties:
 - Name—info
 - Send to machine—AGENT
 - Log name—ELOG
 - Type—INFO
 - Event ID—Less than or equal to (\leq) 4000
 - Monitor mode—CONTINUOUS
 - Description—started
 - Source—LLDSAPNT223
3. Commit the job.

Example: Monitor an Application Log that Occurs on or after a Specified Date

The following job monitors an application log that occurs any time on or after January 11, 2009, 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.

To monitor an application log that occurs on or after a specified date

1. Create a Windows Event Log Monitoring job.
2. Enter the following properties:
 - Name—jan11
 - Send to machine—AGENT
 - Log name—Application
 - Type—INFO
 - Start date/time—20090111 06:30:00
 - Source—LLDSAPNT223
3. Commit the job.

Windows Event Log Monitoring Properties

The Windows Event Log Monitoring category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Category

(Optional) Specifies the event category as displayed in the Windows Event Viewer.

Limits: Up to 256 characters; case-sensitive

JIL attribute: win_event_category

Computer

(Optional) Specifies the local computer name where the event log is monitored.

Limits: Up to 80 characters; case-sensitive

JIL attribute: win_event_computer

Description

(Optional) Specifies the event description as displayed in the Windows Event Viewer.

Limits: Up to 256 characters; case-sensitive

JIL attribute: win_event_description

Event ID

(Optional) Specifies an operator to compare against the value of a Windows Event ID, as follows:

- Greater than (>)
- Less than (<)
- Less than or equal to (<=)
- Equal to (=)
- Greater than or equal to (>=)

Default: Equal to (=)

Windows Event ID

Specifies the Windows event ID to monitor.

Limits: Up to 10 digits

Example: 4000

JIL attributes: win_event_op, win_event_id

Log name

Specifies the name of the event log.

Limits: Up to 256 characters; case-sensitive

JIL attribute: win_log_name

Monitor mode

(Defaulted) Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately.

CONTINUOUS

Monitors for the conditions continuously. Each time the specified conditions occur, an alert is written to the log file.

Note: To end continuous monitoring, you must complete the job manually.

NOW

Checks for the conditions immediately. If the conditions are met, the job completes successfully. If the conditions are not met, the job fails.

WAIT

Waits for the conditions to occur. When the conditions are met, the job completes.

Default: WAIT

JIL attribute: monitor_mode

Source

(Optional) Specifies the event source as displayed in the Windows Event Viewer.

Limits: Up to 256 characters; case-sensitive

JIL attribute: win_event_source

Start date/time

(Optional) Specifies the date and time of the Windows event log. This property uses the following format:

yyyymmdd hh:mm:ss

Limits: Up to 20 characters

Example: 20090501 06:30:00

JIL attribute: win_event_datetime

Type

(Defaulted) Specifies the event type to monitor in the Windows event log.

AUDITF

Specifies the Failure Audit event type.

AUDITS

Specifies the Success Audit event type.

ERROR

Specifies the Error event type.

INFO

Specifies the Information event type.

WARN

Specifies the Warning event type.

Default: ERROR

JIL attribute: win_event_type

Define a Windows Service Monitoring Job

You can define a Windows Service Monitoring job to monitor services on a local Windows computer.

Note: To run these jobs, your system requires CA WA Agent for Windows.

To define a Windows Service Monitoring job

1. Create a Windows Service Monitoring (OMS) job in either Quick Edit or Application Editor.

The Properties section for the Windows Service Monitoring job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Service name

Specifies the name of the local Windows service to be monitored.

3. (Optional) Specify optional [Windows Service Monitoring properties](#) (see page 293).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Windows Service Monitoring job is defined.

More information:

[Windows Service Monitoring Job Examples](#) (see page 293)

Windows Service Monitoring Job Examples

The following example describes a Windows Service Monitoring job.

Example: Monitor a Service for a Status of RUNNING

In this example, the job monitors the service Event Log for a status of RUNNING. The job waits until this status occurs before it completes.

To monitor a service for a status of RUNNING

1. Create a Windows Service Monitoring job.
2. Enter the following properties:
 - Name—running
 - Send to machine—AGENT
 - Service name—Event Log
 - Service status—RUNNING
3. Commit the job.

Windows Service Monitoring Properties

The Windows Service Monitoring category includes the following properties:

Agent Job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Monitor mode

(Defaulted) Specifies whether the job waits until the monitor conditions are met or tries to verify them immediately. The following values are available:

NOW

Checks for the conditions immediately. If the conditions are met, the job completes successfully. If the conditions are not met, the job fails.

WAIT

Waits for the conditions to occur. When the conditions are met, the job completes.

Note: When using the WAIT value, you can specify a lower boundary but you cannot specify an upper boundary.

Default: NOW

JIL attribute: monitor_mode

Service name

Specifies the name of the local Windows service to be monitored.

Limits: Up to 256 characters; case-sensitive

Note: You can specify the service name only or the full path to the service executable.

JIL attribute: win_service_name

Service status

(Defaulted) Specifies the status of the Windows service to be monitored. The following values are available:

CONTINUE PENDING

Specifies that the job monitors the Windows service for a continue pending status.

EXISTS

Specifies that the job checks whether the Windows service exists.

NOT EXISTS

Specifies that the job checks whether the Windows service does not exist.

PAUSED

Specifies that the job monitors the Windows service for a paused status.

PAUSE PENDING

Specifies that the job monitors the Windows service for a pause pending status.

RUNNING

Specifies that the job monitors the Windows service for a running status.

START PENDING

Specifies that the job monitors the Windows service for a start pending status.

STOPPED

Specifies that the job monitors the Windows service for a stopped status.

STOP PENDING

Specifies that the job monitors the Windows service for a stop pending status.

Default: RUNNING

JIL attribute: win_service_status

Chapter 15: Oracle E-Business Suite Jobs

This section contains the following topics:

[Oracle E-Business Suite Jobs](#) (see page 297)

[Define an Oracle E-Business Suite Copy Single Request Job](#) (see page 298)

[Define an Oracle E-Business Suite Request Set Job](#) (see page 301)

[Define an Oracle Applications Single Request Job](#) (see page 310)

Oracle E-Business Suite Jobs

Oracle E-Business Suite jobs let you 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:

Oracle E-Business Suite Single Request

Runs a single program in an Oracle Applications application.

Oracle E-Business Suite Copy Single Request Job

Copies an existing single request defined on Oracle Applications and runs it under the agent.

Oracle E-Business Suite Request Set

Runs multiple programs 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 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.

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. Create an Oracle E-Business Suite Copy Single Request (OACOPY) job in either Quick Edit or Application Editor.

The Properties section for the Oracle E-Business Suite Copy Single Request job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Request ID

Specifies the request ID of the single request you want to copy.

3. Specify the following properties:

Responsibility

Specifies an Oracle E-Business Suite responsibility name.

Application user

Specifies the Oracle E-Business Suite user name that the job runs under.

Note: Your agent administrator can also set default values for these properties using the oa.default.responsibility and oa.default.user parameters.

4. (Optional) Specify optional [Oracle E-Business Suite Copy Single Request properties](#) (see page 312).
5. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
6. Commit the job.

The Oracle E-Business Suite Copy Single Request job is defined.

More information:

[Oracle E-Business Suite Copy Single Request Properties](#) (see page 300)

Oracle E-Business Suite Copy Single Request Job Example

The following example describes a sample Oracle E-Business Suite Copy Single Request job.

Example: Copy a Single Request

Suppose that you want to copy the Oracle E-Business Suite single request with request ID 2255470 and override the Oracle E-Business Suite user name and responsibility name. You also want to monitor the children of the single request program 60 seconds after the program completes.

To copy a single request

1. Create an Oracle E-Business Suite Copy Single Request job.
2. Enter the following properties:
 - Name—REQ_2255470
 - Send to machine—CYB0A
 - Request ID—2255470
 - Application user—SYSADMIN
 - Responsibility—System Administrator
 - Monitor children—selected
 - Monitor children delay—60
3. Commit the job.

Oracle E-Business Suite Copy Single Request Properties

The Oracle E-Business Suite Copy Single Request category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Application user

(Optional) Specifies the Oracle E-Business Suite user name that the job runs under.

Limits: Up to 30 characters; case-sensitive

Notes:

- You must also specify the Responsibility property if you specify the Application user.
- Your agent administrator can set a default value for this property using the oa.default.user parameter.

JIL attribute: oracle_user

Monitor children

(Optional) Indicates whether the children of the Oracle E-Business Suite programs are monitored. When selected, the agent monitors the children. Program children are programs that are released by the parent program.

Note: You must also specify the Monitor children delay property if you specify the Monitor children.

JIL attribute: oracle_mon_children

Monitor children delay

(Optional) Defines the number of seconds to wait after an Oracle E-Business Suite program completes before monitoring its children.

Limits: 0-99999

Note: You must also specify the Monitor children property if you specify the Monitor children delay.

JIL attribute: oracle_mon_children_delay

Request ID

Specifies the request ID of the single request you want to copy. In Oracle Applications, the request ID is found in the Request ID column of the Requests dialog.

Limits: Up to 20 numeric digits

Example: 49389545970965675096795

JIL attribute: request_id

Responsibility

(Optional) Specifies an Oracle E-Business Suite responsibility name.

Limits: Up to 80 characters; case-sensitive

Notes:

- You must also specify the Application user property if you specify the Responsibility.
- Your agent administrator can set a default value for this property using the oa.default.responsibility parameter.

JIL attribute: oracle_resp

Define an Oracle E-Business Suite Request Set Job

You can define an Oracle E-Business Suite Request Set job to run multiple programs in an Oracle E-Business Suite application.

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. Create an Oracle E-Business Suite Request Set (OASET) job in either Quick Edit or Application Editor.

The Properties section for the Oracle E-Business Suite Request Set job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Request set

Specifies the short name of the Oracle E-Business Suite request set.

Application name

Specifies the short name or display name of the Oracle E-Business Suite application that the request set belongs to.

3. Specify the following properties:

Responsibility

Specifies an Oracle E-Business Suite responsibility name.

Application user

Specifies the Oracle E-Business Suite user name that the job runs under.

Note: Your agent administrator can also set default values for these properties using the oa.default.responsibility and oa.default.user parameters.

4. (Optional) Specify optional [Oracle E-Business Suite Request Set properties](#) (see page 306).
5. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
6. Commit the job.

The Oracle E-Business Suite Request Set job is defined.

More information:

[Specify Data for an Individual Program in a Request Set](#) (see page 303)

[Oracle E-Business Suite Request Set Properties](#) (see page 306)

Specify Data for an Individual Program in a Request Set

When you define an Oracle E-Business Suite Request Set job, you can specify data such as arguments, the number of copies to print, and whether to save the output for an individual program in the request set. This data overrides the arguments and print parameters specified for the entire request set.

To specify data for an individual program in a request set

1. Define an Oracle E-Business Suite Request Set job.
2. Select Add in the Program data list field.
A row appears to specify a program in the request set.
3. Specify the order number of the program in the following field:

Number

Specifies a number to identify the program sequence. This number should be a unique integer greater than zero.

JIL attribute: oracle_programdata: index

4. Specify one or more values in the following fields:

Arguments

Defines the argument values to override the default values that are defined by the registered Oracle E-Business Suite Concurrent Manager program.

Note: Separate each argument with a comma. Do not add a space after a comma unless the argument value starts with a space. If a value starts with a space, enclose the value in single quotation marks.

Limits: Up to 500 characters

JIL attribute: oracle_programdata: args

Printer

Specifies the name of a printer that Oracle E-Business Suite can print to. This printer must be defined in Oracle E-Business Suite. In Oracle E-Business Suite, the printer name is specified as a request definition option and is found in the Printer column of the Upon Completion dialog.

Note: Enclose paths that contain delimiters (such as spaces) in double quotation marks.

Limits: Up to 80 characters

JIL attribute: oracle_programdata: printer

Print style

Specifies a print style. The value must match the name of an Oracle E-Business Suite print style. In Oracle E-Business Suite, the print style is specified as a request definition option and is found in the Style field of the Upon Completion dialog.

Limits: Up to 80 characters

JIL attribute: oracle_programdata: print_style

Print copies

Specifies the number of copies to print. In Oracle E-Business Suite, the number of copies is specified as a request definition option and is found in the Copies column of the Upon Completion dialog.

Limits: 0-999

JIL attribute: oracle_programdata: print_copies

Save output

Indicates whether to save output from the job execution. When yes is selected, the agent saves output from an Oracle E-Business Suite program.

JIL attribute: oracle_programdata: saveop

These properties override the arguments and print parameters specified for the entire request set.

Example: Specify Values for Individual Programs in a Request Set

Suppose that you want to define an Oracle E-Business Suite Request Set job to run a 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 Q8.

The first program data list identifies the first program in the request set. Subsequent statements override the default values for this program. The first program uses the arguments T23 and R1 and prints two copies using the Q1 printer in PORTRAIT style.

The second program data list identifies the fifth program in the request set. Subsequent statements override the default values for this program. The fifth program uses arguments R and R1 and prints three copies using the Q2 printer in PORTRAIT style.

The other programs in the request set use the default values.

To specify values for individual programs in a request set

1. Create an Oracle E-Business Suite Request Set job.
2. Enter the following properties:
 - Name—oaset_resp
 - Send to machine—oaagent
 - Request set—EXTRACTS
 - Application name—Application Object Library
 - Application name type—DISPLAY
 - Application user—SYSADMIN
 - Responsibility—System Administrator
 - Printer—Q8
 - Print style—LANDSCAPE
 - Print copies—1
3. Add a program data list with the following values:
 - Number—1
 - Arguments—T23,R1
 - Printer—Q1
 - Print style—PORTRAIT
 - Print copies—2
 - Save output—selected
4. Add another program data list with the following values:
 - Number—5
 - Arguments—R,R1
 - Printer—Q2
 - Print style—PORTRAIT
 - Print copies—3
 - Save output—unselected
5. Commit the job.

Oracle E-Business Suite Request Set Job Example

The following example describes a sample Oracle E-Business Suite Request Set job.

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 Applications. The job uses the default Oracle E-Business Suite responsibility name and user name defined on the agent.

To run a request set

1. Create an Oracle E-Business Suite Request Set job.
2. Enter the following properties:
 - Name—oaset_resp
 - Send to machine—oaagent
 - Request set—FNDRSSUB1310
 - Application name—BIS
 - Application name type—SHORT
 - Application user—SYSADMIN
 - Responsibility—System Administrator
3. Commit the job.

Oracle E-Business Suite Request Set Properties

The Oracle E-Business Suite Request Set category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Application name

Specifies the short name or display name of the Oracle E-Business Suite application that the request set belongs to.

Limits: Up to 256 characters

JIL attribute: oracle_appl_name

Application name type

(Defaulted) Specifies whether the name of the Oracle E-Business Suite application is the short name or the display name, as follows:

- **SHORT**—The short name is defined by the Oracle E-Business Suite Concurrent Manager (ACM).
- **DISPLAY**—The display name is part of the request definition in Oracle E-Business Suite and is found in the Application field.

Default: SHORT

JIL attribute: oracle_appl_name_type

Application user

(Optional) Specifies the Oracle E-Business Suite user name that the job runs under.

Limits: Up to 30 characters; case-sensitive

Notes:

- You must also specify the Responsibility property if you specify the Application user.
- Your agent administrator can set a default value for this property using the oa.default.user parameter.

JIL attribute: oracle_user

Monitor children

(Optional) Indicates whether the children of the Oracle E-Business Suite programs are monitored. When selected, the agent monitors the children. Program children are programs that are released by the parent program.

Note: You must also specify the Monitor children delay property if you specify the Monitor children.

JIL attribute: oracle_mon_children

Monitor children delay

(Optional) Defines the number of seconds to wait after an Oracle E-Business Suite program completes before monitoring its children.

Limits: 0-99999

Note: You must also specify the Monitor children property if you specify the Monitor children delay.

JIL attribute: oracle_mon_children_delay

Print copies

(Optional) Specifies the number of copies to print. In Oracle E-Business Suite, the number of copies is specified as a request definition option and is found in the Copies column of the Upon Completion dialog.

Limits: 0-999

JIL attribute: oracle_print_copies

Print style

(Optional) Specifies a print style. The value must match the name of an Oracle E-Business Suite print style. In Oracle E-Business Suite, the print style is specified as a request definition option and is found in the Style field of the Upon Completion dialog.

Limits: Up to 80 characters

Example: PORTRAIT

JIL attribute: oracle_print_style

Printer

(Optional) Specifies the name of a printer that Oracle E-Business Suite can print to. This printer must be defined in Oracle E-Business Suite. In Oracle E-Business Suite, the printer name is specified as a request definition option and is found in the Printer column of the Upon Completion dialog.

Limits: Up to 80 characters

Example: "\\printer path\K6700"

Note: Enclose paths that contain delimiters (such as spaces) in double quotation marks.

JIL attribute: oracle_printer

Program data list

(Optional) Specifies [data for an individual program](#) (see page 303) in an Oracle E-Business Suite request set.

Limits: Up to 4096 characters

Note: To add program data, select Add and click Go. A row of data fields appears below the Program data list field.

JIL attribute: oracle_programdata

Request set

Specifies the short name of the Oracle E-Business Suite request set. In Oracle E-Business Suite, the request set short name is part of the request set definition and is found in the Set Code field of the Request Set dialog.

Limits: Up to 256 characters

JIL attribute: oracle_req_set

Responsibility

(Optional) Specifies an Oracle E-Business Suite responsibility name.

Limits: Up to 80 characters; case-sensitive

Notes:

- You must also specify the Application user property if you specify the Responsibility.
- Your agent administrator can set a default value for this property using the oa.default.responsibility parameter.

JIL attribute: oracle_resp

Save output

(Optional) Indicates whether to save output from the job execution. When YES is selected, the agent saves output from an Oracle E-Business Suite program.

JIL attribute: oracle_save_output

Use argument defaults

(Optional) Indicates whether the agent uses default values for arguments. When selected, the agent uses default argument values.

Note: The default arguments are defined in Oracle E-Business Suite.

JIL attribute: oracle_use_arg_def

Define an Oracle Applications Single Request Job

You can define an Oracle E-Business Suite Single Request job to run a single program in an Oracle Applications application.

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. Create an Oracle E-Business Suite Single Request job in either Quick Edit or Application Editor.

The Properties section for the Oracle E-Business Suite Single Request job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Program name

Specifies the short name of the Oracle E-Business Suite single request program.

Application name

Specifies the short name or display name of the Oracle E-Business Suite application that the request set belongs to.

3. Specify the following properties:

Responsibility

Specifies an Oracle E-Business Suite responsibility name.

Application user

Specifies the Oracle E-Business Suite user name that the job runs under.

Note: Your agent administrator can also set default values for these properties using the oa.default.responsibility and oa.default.user parameters.

4. (Optional) Specify optional [Oracle E-Business Suite Single Request properties](#) (see page 312).
5. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
6. Commit the job.

The Oracle E-Business Suite Single Request job is defined.

More information:

[Oracle E-Business Suite Single Request Properties](#) (see page 312)

Oracle E-Business Suite Single Request Job Example

The following example describes a sample Oracle E-Business Suite Single Request job.

Example: Run an Oracle E-Business Suite Single Request

In this example, the Oracle E-Business Suite Single Request job runs the single request XXCOFI on the Oracle E-Business Suite system. The single request belongs to the application with the display name Application Object Library in Oracle E-Business Suite.

To run an Oracle E-Business Suite single request

1. Create an Oracle E-Business Suite Single Request job.
2. Enter the following properties:
 - Name—REQ_XXCOFI
 - Send to machine—CYB0A
 - Program name—XXCOFI
 - Application name—'Application Object Library'
 - Application name type—DISPLAY
 - Application user—SYSADMIN
 - Responsibility—System Administrator
3. Commit the job.

Oracle E-Business Suite Single Request Properties

The Oracle E-Business Suite Single Request category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Application name

Specifies the short name or display name of the Oracle E-Business Suite application that the request set belongs to.

Limits: Up to 256 characters

JIL attribute: oracle_appl_name

Application name type

(Defaulted) Specifies whether the name of the Oracle E-Business Suite application is the short name or the display name, as follows:

- **SHORT**—The short name is defined by the Oracle E-Business Suite Concurrent Manager (ACM).
- **DISPLAY**—The display name is part of the request definition in Oracle E-Business Suite and is found in the Application field.

Default: SHORT

JIL attribute: oracle_appl_name_type

Application user

(Optional) Specifies the Oracle E-Business Suite user name that the job runs under.

Limits: Up to 30 characters; case-sensitive

Notes:

- You must also specify the Responsibility property if you specify the Application user.
- Your agent administrator can set a default value for this property using the oa.default.user parameter.

JIL attribute: oracle_user

Monitor children

(Optional) Indicates whether the children of the Oracle E-Business Suite programs are monitored. When selected, the agent monitors children. Program children are programs that are released by the parent program.

Note: You must also specify the Monitor children delay property if you specify the Monitor children.

JIL attribute: oracle_mon_children

Monitor children delay

(Optional) Defines the number of seconds to wait after an Oracle E-Business Suite program completes before monitoring its children.

Limits: 0-99999

Note: You must also specify the Monitor children property if you specify the Monitor children delay.

JIL attribute: oracle_mon_children_delay

Print copies

(Optional) Specifies the number of copies to print. In Oracle E-Business Suite, the number of copies is specified as a request definition option and is found in the Copies column of the Upon Completion dialog.

Limits: 0-999

JIL attribute: oracle_print_copies

Print style

(Optional) Specifies a print style. The value must match the name of an Oracle E-Business Suite print style. In Oracle E-Business Suite, the print style is specified as a request definition option and is found in the Style field of the Upon Completion dialog.

Limits: Up to 80 characters

Example: PORTRAIT

JIL attribute: oracle_print_style

Printer

(Optional) Specifies the name of a printer that Oracle E-Business Suite can print to. This printer must be defined in Oracle E-Business Suite. In Oracle E-Business Suite, the printer name is specified as a request definition option and is found in the Printer column of the Upon Completion dialog.

Limits: Up to 80 characters

Example: "\\printer path\K6700"

Note: Enclose paths that contain delimiters (such as spaces) in double quotation marks.

JIL attribute: oracle_printer

Program arguments

(Optional) Defines each argument value to pass to an Oracle E-Business Suite single request. In Oracle E-Business Suite, arguments are part of the program definition and are found in the Concurrent Program Parameters dialog.

Limits: Up to 500 characters

Notes:

- You can specify up to 100 arguments.
- Separate each argument with a comma.
- Do not add a space after a comma unless the argument value starts with a space. If a value starts with a space, enclose the value in single quotation marks.

JIL attribute: oracle_args

Program name

Specifies the short name of the Oracle E-Business Suite single request program. The program short name must be registered in the Oracle E-Business Suite Concurrent Manager. In Oracle E-Business Suite, the program short name is part of the program definition and is found in the Short Name field of the Concurrent Programs dialog.

Limits: Up to 256 characters

JIL attribute: oracle_program

Request description

(Optional) Defines a description for an Oracle E-Business Suite Single Request job. The description displays in the Name column of the Requests dialog.

Limits: Up to 256 characters

JIL attribute: oracle_desc

Responsibility

(Optional) Specifies an Oracle E-Business Suite responsibility name.

Limits: Up to 80 characters; case-sensitive

Notes:

- You must also specify the Application user property if you specify the Responsibility.
- Your agent administrator can set a default value for this property using the oa.default.responsibility parameter.

JIL attribute: oracle_resp

Save output

(Optional) Indicates whether to save output from the job execution. When selected, the agent saves output from an Oracle E-Business Suite program.

JIL attribute: oracle_save_output

Use argument defaults

(Optional) Indicates whether the agent uses default values for arguments. The default arguments are defined in Oracle E-Business Suite. When selected, the agent uses default argument values.

JIL attribute: oracle_use_arg_def

Chapter 16: SAP Jobs

This section contains the following topics:

- [SAP Jobs \(see page 318\)](#)
- [How to Use Filters to Define SAP Jobs \(see page 319\)](#)
- [Define an SAP Batch Input Session Job \(see page 324\)](#)
- [Define an SAP Business Warehouse InfoPackage Job \(see page 329\)](#)
- [Define an SAP Business Warehouse Process Chain Job \(see page 333\)](#)
- [Define an SAP Data Archiving Job \(see page 336\)](#)
- [Define an SAP Event Monitor Job \(see page 346\)](#)
- [Define an SAP Job Copy Job \(see page 350\)](#)
- [Define an SAP Process Monitor Job \(see page 354\)](#)
- [Using Success and Failure Messages within an SAP Job \(see page 356\)](#)
- [SAP Process Monitor Properties \(see page 357\)](#)
- [Define an SAP R/3 Job \(see page 360\)](#)
- [Define Recipients for the Job's Output \(see page 367\)](#)
- [Define Print or Archive Parameters for a Step \(see page 370\)](#)
- [Send the SAP Spool File by Email on Step Completion or Failure \(see page 380\)](#)

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.

How to Use Filters to Define SAP Jobs

You can filter and list jobs running on your SAP system, copy them to your job flow, and schedule them on the CA Workload Automation AE server. Reusing jobs saves time and effort. You can reuse existing jobs and modify their properties as required instead of defining the properties of new jobs from scratch.

Note: Filtering capability is only available for SAP R/3, SAP Batch Input Session, SAP Business Warehouse InfoPackage, and SAP Business Warehouse Process Chain jobs.

To use filters to define SAP jobs, follow these steps:

1. [Create an SAP filter](#) (see page 320).
2. [Display a list of jobs on the SAP system](#) (see page 322).
3. [Copy a job from the SAP system to a job flow](#) (see page 323).

Create an SAP Filter

Your SAP system may contain thousands of individual jobs. You can save time by creating filters for jobs you search for and work with frequently. For example, you might want to create filters for jobs with specific names created by specific users. When you use the filter, the SAP jobs that match its criteria display in the Results for SAP Filter Criteria pane. Using a job filter also lets your system return the requested jobs more quickly and with a lower probability of disturbing system performance.

You can create separate filters for SAP R/3, SAP Batch Input Session, SAP Business Warehouse InfoPackage, and SAP Business Warehouse Process Chain jobs.

Note: To display the SAP Filters and SAP Filter Results panes, select one or more of the following SAP job types in the Customize dialog: SAP R/3, SAP Batch Input Session, SAP BW Process Chain, or SAP BW InfoPackage. Additionally, verify that Hide SAP Drag and Drop Panes is not checked in the Customize dialog.

To create an SAP filter

1. Click Add in the SAP Filters pane.
The Create New SAP Filter dialog opens.
2. Specify the following required fields to identify the filter:

Filter type

Specifies the type of SAP job the filter will display.

Filter name

Specifies a name for the filter.

Limits: Up to 80 characters

3. Specify the following logon parameters:

Server

Specifies the server on which to perform the search.

Machine

Specifies the name of the computer where the agent runs.

Owner

(Optional) Specifies the user ID to use when running the job. The value must be a valid user ID.

Limits: Up to 145 characters

Client

(Optional) Specifies the client within the SAP system.

Limits: Up to three digits

Language

(Optional) Specifies a character code representing a valid language for SAP.

Limits: Up to two characters; case-sensitive

RFC destination

(Optional) Indicates the destination specified in the connection properties file during installation.

Limits: Up to 256 characters; case-sensitive

4. Specify the following required filter criteria:

SAP job name

Specifies the SAP job name. You can use an asterisk (*) as a wildcard character to represent multiple characters.

Note: This field applies to SAP R/3 and SAP Batch Input Session filter types only.

SAP user name

Specifies the SAP user name. You can use an asterisk (*) as a wildcard character to represent multiple characters.

Note: This field applies to SAP R/3 and SAP Batch Input Session filter types only.

5. (Optional) Specify optional filter criteria to narrow the job search.

6. Click Save.

The SAP filter is added to the SAP Filters list.

Display a List of Jobs on the SAP System

You can display a list of jobs on the SAP system that match the filter criteria you have defined.

Note: To display the SAP Filters and SAP Filter Results panes, select one or more of the following SAP job types in the Customize dialog: SAP R/3, SAP Batch Input Session, SAP BW Process Chain, or SAP BW InfoPackage. Additionally, verify that Hide SAP Drag and Drop Panes is not checked in the Customize dialog.

To display a list of jobs on the SAP system

Select the SAP filter in the SAP Filters pane.

The jobs found on the SAP system are listed in the Results for SAP Filter Criteria pane.

Note: If a very large number of jobs that match the search criteria are found, the FTP Credentials dialog opens. You must enter your credentials to continue.

More information:

[Create an SAP Filter](#) (see page 320)

Copy a Job from the SAP System to a Job Flow

After you display a list of jobs on the SAP system that match your filter criteria, you can select each job you want to reuse. You can then copy each job to the job flow, and update the job definition as required.

To copy a job from the SAP system to a job flow

1. [Display a list of jobs on the SAP system](#) (see page 322).
A list of SAP jobs that satisfy your filter is generated.
2. Locate the job you want to copy to the job flow.
3. Drag and drop the job onto the workspace.
The job icon appears on the workspace.
4. Right-click the job icon, and select Edit.
The properties section for the job opens in the lower half of Application Editor.
5. Modify the properties as required.
6. Click Apply.
The job is copied to the job flow.

Define an SAP Batch Input Session Job

You can define an SAP Batch Input Session job to import large amounts of data from external systems to the SAP system. You create Batch Input Session jobs on the SAP system. This procedure lets you create a job from scratch; however, you can use an SAP filter and copy an existing job from the SAP system to save time.

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. Create an SAP Batch Input Session (SAPBDC) job in either Quick Edit or Application Editor.

The Properties section for the SAP Batch Input Session job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

SAP job name

Specifies the name of the ABAP job that creates the BDC session on the SAP system.

3. (Optional) Specify optional [SAP Batch Input Session properties](#) (see page 325).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The SAP Batch Input Session job is defined.

More information:

[Copy a Job from the SAP System to a Job Flow](#) (see page 323)

[Define Recipients for the Job's Output](#) (see page 367)

[Define Print or Archive Parameters for a Step](#) (see page 370)

[Send the SAP Spool File by Email on Step Completion or Failure](#) (see page 380)

[SAP Batch Input Session Job Example](#) (see page 325)

[Using Success and Failure Messages within an SAP Job](#) (see page 356)

SAP Batch Input Session Job Example

The following example describes a sample SAP Batch Input Session job.

Example: Define an SAP Batch Input Session Job

This example runs the ZBDCTEST ABAP that creates a Batch Input Session (BDC ABAP) on the SAP system. The job runs as soon as possible after the job is defined. After this job runs, the BDC job starts the data transfer.

To define an SAP Batch Input Session job

1. Create an SAP Batch Input Session job.
2. Enter the following properties:
 - Name—bdcjob
 - Send to machine—sapagent
 - Owner—WAAESAP@sapagent
 - SAP job name—ZBDCTEST
 - Release option—As soon as possible
3. Open the Step Parameters dialog.
4. Enter the following property on the Command tab:
 - ABAP Name—ZBDCTEST
5. Commit the job.

SAP Batch Input Session Properties

The SAP Batch Input Session category includes the following properties:

Agent job class

Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Error rate (%)

(Optional) Specifies the maximum acceptable error rate as a percentage of the total transactions. When set to 100, all errors are acceptable. When set to zero (0), the job fails if any transaction contains an error. When set to 5, for example, the job fails if more than five percent of the transactions contain errors.

Limits: 0-100

Default: 0

JIL attribute: bdc_err_rate

Extended logging

(Optional) Indicates whether to generate advanced logging of the Batch Input Session (BDC) running on the SAP system. When selected, the agent uses extended logging.

JIL attribute: bdc_ext_log

Logon Parameters: Client

(Optional) Specifies the client within the SAP system. The value corresponds to the General data, Target server field on the Define Background Job dialog.

Limits: Up to three digits

JIL attribute: sap_client

Logon Parameters: Language

(Optional) Specifies a character code representing a valid language for SAP.

Limits: Up to two characters; case-sensitive

JIL attribute: sap_lang

Logon Parameters: RFC destination

(Optional) Indicates the destination specified in the connection properties file during installation.

Limits: Up to 256 characters; case-sensitive

JIL attribute: sap_rfc_dest

Office output

(Optional) Indicates whether to save outgoing documents to the SAPoffice outbox of the SAP user associated with the job. When selected, the agent saves outgoing documents to the SAPoffice outbox.

JIL attribute: sap_office

Processed rate (%)

(Optional) Specifies the minimum acceptable process rate as a percentage of the total transactions. When set to 100, the job fails if all transactions are not processed. When set to zero (0), no minimum processed transactions are required. When set to 90, for example, the job fails if less than 90 percent of the transactions are processed.

Limits: 0-100

JIL attribute: bdc_proc_rate

Release option

(Defaulted) Specifies the action to take with a job after it is defined, as follows:

- Immediately—Releases the job immediately
- As soon as possible—Releases the job as soon as possible

Default: As soon as possible

JIL attribute: sap_release_option

SAP job class

(Optional) Specifies a valid SAP system job class.

Limits: Only one character

JIL attribute: sap_job_class

SAP job name

Specifies the name of the ABAP job that creates the BDC session on the SAP system.

Limits: Up to 32 characters; case-sensitive

JIL attribute: sap_job_name

Spool list recipients

(Optional) Specifies the [recipient properties](#) (see page 369) for the job's output (spool file).

Limits: Up to 4096 characters

Note: To add spool list recipients, select Add and click Go. A row appears below the Spool list recipients field. Click to open the Spool List Recipient dialog to specify the recipients.

JIL attribute: sap_recipients

Step parameters

(Optional) Specifies the [SAP R/3 step specifications](#) (see page 372).

Limits: Up to 4096 characters

Note: The SAP Batch Input Session job can only have one step. Click  to open the Step Parameters dialog to specify the parameters for the job definition.

JIL attribute: sap_step_parms

System

(Optional) Specifies the system the BDC session should be executed on.

Limits: Up to 256 characters

JIL attribute: bdc_system

Target system

(Optional) Specifies the host computer within the SAP system architecture that is capable of running SAP system workload.

Limits: Up to 256 characters; case-sensitive

Note: You can click  to open the Browse SAP Target Systems dialog to locate the host computer.

JIL attribute: sap_target_sys

Define an SAP Business Warehouse InfoPackage Job

You can define an SAP Business Warehouse InfoPackage job to transfer data from any data source into an SAP Business Warehouse system. You use this procedure to create a new job. You can also copy an existing job from the SAP system to save time.

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 Business Warehouse InfoPackage job

1. Create an SAP BW InfoPackage (SAPBWIP) job in either Quick Edit or Application Editor.

The Properties section for the SAP BW InfoPackage job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Info package name

Specifies the name of the Business Warehouse InfoPackage.

Note: You can click  to open the Search SAP Info Packages dialog to locate an SAP InfoPackage for the job definition.

3. (Optional) Specify optional [SAP BW InfoPackage properties](#) (see page 330).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The SAP BW InfoPackage job is defined.

More information:

[Copy a Job from the SAP System to a Job Flow](#) (see page 323)

[SAP BW InfoPackage Job Examples](#) (see page 330)

SAP BW InfoPackage Job Examples

The following examples describe sample SAP BW InfoPackage jobs.

Example: Define an SAP BW InfoPackage Job

This example runs the Business Warehouse InfoPackage 0PAK_D2XZMZ1HD5WFVFL3EN1NVIT4V in the SAP destination SM1.

To define an SAP BW InfoPackage job

1. Create an SAP BW InfoPackage job.
2. Enter the following properties:
 - Name—InfoBW2
 - Send to machine—sapagent
 - Owner—WAAESAP@sapagent
 - Logon Parameters: RFC destination—SM1
 - SAP job name—InfoBW2
 - Info package name—0PAK_D2XZMZ1HD5WFVFL3EN1NVIT4V
3. Commit the job.

SAP BW InfoPackage Properties

The SAP BW InfoPackage category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Info package details

(Optional) Specifies the SAP Business Warehouse (BW) InfoPackage data selection criteria, as follows:

Field Name

Specifies the BW InfoPackage Technical Name.

Limits: Up to 100 characters

High

Specifies the BW InfoPackage Technical To Value.

Limits: Up to 256 characters

Low

Specifies the BW InfoPackage Technical From Value.

Limits: Up to 256 characters

Object Name

Specifies the BW InfoPackage Technical InfoObject.

Limits: Up to 100 characters

Note: To assign no value to this data selection criterion, use single quotation marks.

Operation

Specifies the BW InfoPackage Technical Operation, as follows:

- BT—Between
- EQ—Equal

Sign

Specifies the BW InfoPackage Technical Sign, as follows:

- E—Exclude
- I—Include

Limits: Up to 4096 characters

Note: You must click  to complete the InfoPackage details. The agent returns values for the InfoPackage based on details that are configured in the SAP system.

JIL attribute: sap_ext_table

Info package name

Specifies the name of the Business Warehouse InfoPackage.

Limits: Up to 30 valid SAP characters; case-sensitive

Note: You can click  to open the Search SAP Info Packages dialog to locate the SAP InfoPackage.

JIL attribute: sap_info_pack

Logon Parameters: Client

(Optional) Specifies the client within the SAP system. The value corresponds to the General data, Target server field on the Define Background Job dialog.

Limits: Up to three digits

JIL attribute: sap_client

Logon Parameters: Language

(Optional) Specifies a character code representing a valid language for SAP.

Limits: Up to two characters; case-sensitive

JIL attribute: sap_lang

Logon Parameters: RFC destination

(Optional) Indicates the destination specified in the connection properties file during installation.

Limits: Up to 256 characters; case-sensitive

JIL attribute: sap_rfc_dest

SAP job name

Specifies the SAP R/3 job name.

Limits: Up to 32 characters; case-sensitive

JIL attribute: sap_job_name

Define an SAP Business Warehouse Process Chain Job

You can define an SAP Business Warehouse Process Chain job to run a sequence of background processes on the SAP system. Some SAP processes trigger events that can start other processes. A job runs the individual processes in the chain as job steps. You use this procedure to create a new job. You can also copy an existing job from the SAP system to save time.

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 Business Warehouse Process Chain job

1. Create an SAP BW Process Chain (SAPBWPC) job in either Quick Edit or Application Editor.

The Properties section for the SAP BW Process Chain job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Chain identifier

Specifies the name of the Business Warehouse Process Chain.

Note: You can click  to open the Get Process Chain Identifier dialog to locate a BW Process Chain for the job definition.

3. (Optional) Specify optional [SAP BW Process Chain properties](#) (see page 334).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The SAP BW Process Chain job is defined.

More information:

[Copy a Job from the SAP System to a Job Flow](#) (see page 323)

[Windows Service Monitoring Job Examples](#) (see page 293)

SAP BW Process Chain Job Example

The following example describes a sample SAP BW Process Chain job.

Example: Define an SAP BW Process Chain Job

This example defines the SAPBWPC3 job, which runs the DEMO_CHAIN1 Process Chain on the SAP system. The language used to log on to the SAP system is English.

To define an SAP BW Process Chain job

1. Create an SAP BW Process Chain job.
2. Enter the following properties:
 - Name—SAPBWPC3
 - Send to machine—sapagent
 - Owner—WAAESAP@sapagent
 - Logon Parameters: Language—EN
 - Logon Parameters: RFC destination—SM1
 - Chain identifier—DEMO_CHAIN1
3. Commit the job.

SAP BW Process Chain Properties

The SAP BW Process Chain category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Chain identifier

Specifies the name of the Business Warehouse Process Chain.

Limits: Up to 30 characters

Note: You can click  to open the Get Process Chain Identifier dialog to locate the process chain.

JIL attribute: sap_chain_id

Logon Parameters: Client

(Optional) Specifies the client within the SAP system. The value corresponds to the General data, Target server field on the Define Background Job dialog.

Limits: Up to three digits

JIL attribute: sap_client

Logon Parameters: Language

(Optional) Specifies a character code representing a valid language for SAP.

Limits: Up to two characters; case-sensitive

JIL attribute: sap_lang

Logon Parameters: RFC destination

(Optional) Indicates the destination specified in the connection properties file during installation.

Limits: Up to 256 characters; case-sensitive

JIL attribute: sap_rfc_dest

Define an SAP Data Archiving Job

You can define an SAP Data Archiving job to store information described in an SAP archiving object in 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. Create an SAP Data Archiving (SAPDA) job in either Quick Edit or Application Editor.
The Properties section for the SAP Data Archiving job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Archiving object name

Specifies the name of the archiving object.

Note: You can click  to open the Search Archiving Objects dialog to locate an archiving object for the job definition.

Archiving object variant

Specifies the name of the archiving object variant.

Note: You can click  to open the Browse Archiving Objects Variants dialog to locate an archiving object variant for the job definition.

Print parameters

Specifies [print parameters](#) (see page 342).

Note: You can click  to open the Print Parameters dialog to specify the parameters for the job definition.

3. (Optional) Specify optional [SAP Data Archiving properties](#) (see page 338).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The SAP Data Archiving job is defined.

More information:

[Archiving Parameters](#) (see page 339)

[Print Parameters](#) (see page 342)

[Define Print or Archive Parameters for a Step](#) (see page 370)

SAP Data Archiving Job Example

The following example describes a sample SAP Data Archiving job.

Example: Define an SAP Data Archiving Job

This example defines a job that stores information described in the BC_ARCHIVE Archiving Object in an SAP data archive. The archiving object variant is BC_ARCVARIANT.

To define an SAP Data Archiving job

1. Create an SAP Data Archiving job.
2. Enter the following properties:
 - Name—SAPDA_job
 - Send to machine—sapagent
 - Owner—WAAESAP@sapagent
 - Archiving object name—BC_ARCHIVE
 - Archiving object variant—BC_ARCVARIANT
3. Open the Print Parameters dialog.
4. Enter the following properties:
 - Archiving Mode—Archive
 - Output Device—LP01
5. Save the job.

SAP Data Archiving Properties

The SAP Data Archiving category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Archiving object name

Specifies the name of the archiving object.

Limits: Up to 256 characters; case-sensitive

Note: You can click  to open the Search Archiving Objects dialog to locate an archiving object for the job definition.

JIL attribute: arc_obj_name

Archiving object variant

Specifies the name of the archiving object variant.

Limits: Up to 256 characters; case-sensitive

Note: You can click  to open the Browse Archiving Objects Variants dialog to locate an archiving object variant for the job definition.

JIL attribute: arc_obj_variant

Archiving parameters

(Optional) Specifies [archiving parameters](#) (see page 339).

Limits: Up to 4096 characters

JIL attribute: arc_parms

Logon Parameters: Client

(Optional) Specifies the client within the SAP system. The value corresponds to the General data, Target server field on the Define Background Job dialog.

Limits: Up to three digits

JIL attribute: sap_client

Logon Parameters: Language

(Optional) Specifies a character code representing a valid language for SAP.

Limits: Up to two characters; case-sensitive

JIL attribute: sap_lang

Logon Parameters: RFC destination

(Optional) Indicates the destination specified in the connection properties file during installation.

Limits: Up to 256 characters; case-sensitive

JIL attribute: sap_rfc_dest

Print parameters

Specifies [print parameters](#) (see page 342).

Limits: Up to 4096 characters

Note: You can click  to open the Print Parameters dialog to specify the parameters for the job definition.

JIL attribute: sap_print_parms

Target system

(Optional) Specifies the host computer within the SAP system architecture that is capable of running SAP system workload.

Limits: Up to 256 characters; case-sensitive

Note: You can click  to open the Browse SAP Target Systems dialog to locate the host computer.

JIL attribute: sap_target_sys

Archiving Parameters

The Archiving Parameters dialog includes the following parameters:

ArchiveLink Host

(Optional) Specifies the SAP archive link RPC host.

Limits: Up to 32 characters; case-sensitive

JIL attribute: arc_host_link

ArchiveLink Information

(Optional) Specifies the archive link information.

Limits: Up to 3 characters; case-sensitive

Note: This parameter is required if the printer parameter Archiving mode is archived or both.

JIL attribute: arc_info

ArchiveLink Object Type

(Optional) Specifies the type of external system archive object. This parameter corresponds to the Obj. type field on the Define Background Archive Parameters dialog.

Limits: Up to 10 characters; case-sensitive

Note: This parameter is required if the printer parameter Archiving mode is archived or both.

JIL attribute: arc_obj_type

Archiving Path

(Optional) Specifies the standard archive path.

Limits: Up to 70 characters; case-sensitive

JIL attribute: arc_path

Client

(Optional) Specifies the archive link client.

JIL attribute: arc_client

Connection Name

(Optional) Specifies the archive link communication connection.

Limits: Up to 14 characters; case-sensitive

JIL attribute: arc_connect

Date

(Optional) Specifies the date to do the archiving in YYYYMMDD format, using the SAP agent time zone.

Limits: Up to 8 characters

JIL attribute: arc_date

Document Class

(Optional) Specifies the archive link document class.

Limits: Up to 20 characters; case-sensitive

JIL attribute: arc_doc_class

Document Type

(Optional) Specifies the name of the document type. This parameter corresponds to the Doc type field on the Define Background Archive Parameters dialog.

Limits: Up to 10 characters; case-sensitive

Note: This parameter is required if the printer parameter Archiving mode is archived or both.

JIL attribute: arc_doc_type

Format

(Optional) Specifies the SAP archive link output format.

Limits: Up to 16 characters; case-sensitive

JIL attribute: arc_format

Printer

(Optional) Specifies the SAP archive link target printer.

Limits: Up to 4 characters; case-sensitive

JIL attribute: arc_printer

Protocol

(Optional) Specifies the archive storage connection protocol.

Limits: Up to 8 characters; case-sensitive

JIL attribute: arc_protocol

Report

(Optional) Specifies the archive link report name.

Limits: Up to 30 characters; case-sensitive

JIL attribute: arc_report

Service Destination

(Optional) Specifies the SAP archive link RPC service/destination.

Limits: Up to 32 characters; case-sensitive

JIL attribute: arc_service

Target Storage System

(Optional) Specifies the archive link target storage system.

Limits: Up to 2 characters; case-sensitive

JIL attribute: arc_storage

Text

(Optional) Specifies the archive link Text Information field.

Limits: Up to 30 characters; case-sensitive

JIL attribute: arc_text

User

(Optional) Specifies the archive data element for user.

Limits: Up to 16 characters

JIL attribute: arc_userid

Version

(Optional) Specifies the archive version.

Limits: Up to 4 characters

JIL attribute: arc_version

Print Parameters

The Print Parameters dialog includes the following parameters:

Archiving Mode

Specifies whether the spool file is printed, archived, or both.

Note: When Print is selected, the Archiving Parameters dialog is disabled.

JIL attribute: prt_arc_mode

Authorization

(Optional) Specifies the password for viewing the print spool list.

Limits: Up to 12 characters; case-sensitive

JIL attribute: authorization

Dataset Name

(Optional) Specifies the print spool data set name.

Limits: Up to 6 characters; case-sensitive

JIL attribute: dataset

Delete After Output

(Optional) Indicates whether the spool request associated with the ABAP is deleted after printing. When selected, the agent deletes the spool request after printing. This parameter corresponds to the SAPGUI Spool options, Delete after output field on the Background Print Parameters dialog.

JIL attribute: release

Department

(Optional) Specifies the department name to print on the cover page.

Limits: Up to 12 characters; case-sensitive

JIL attribute: dept

Format

(Optional) Specifies an output format for an SAP report.

Limits: Up to 64 characters; case-sensitive

Note: You can click  to open the Get Print Format dialog to locate the output format for the report.

JIL attribute: format

New Spool Request

(Optional) Indicates whether to create a new spool request or append the spool request to an existing spool request with similar attributes (if any). When selected, the agent creates a new spool request.

JIL attribute: new_spool

Number of Columns

(Optional) Specifies the number of columns to be printed on the page.

JIL attribute: num_columns

Number of Copies

(Optional) Specifies the number of print copies.

JIL attribute: copies

Number of Lines

(Optional) Specifies the number of lines to be printed on the page.

JIL attribute: num_lines

OS Cover Page

(Optional) Specifies whether to print a host page, as follows:

- Y—Prints the host page
- N—Does not print the host page
- D—Uses the SAP default

JIL attribute: host_page

Output Device

Specifies the output device name. This parameter corresponds to the SAPGUI Output device field on the Background Print Parameters dialog.

Limits: Up to 16 characters; case-sensitive

Note: You can click  to open the Get Output Devices dialog to locate the output device.

JIL attribute: dest

Print Immediately

(Optional) Indicates whether to print the job immediately after completion. When selected, the agent prints the job immediately. This parameter corresponds to the SAPGUI Spool options, Print immediately field on the Background Print Parameters dialog.

JIL attribute: print_imm

Print Cover Page

(Optional) Indicates whether to include an SAP cover page with the report output. When selected, the agent prints the report with a cover page.

JIL attribute: banner_page

Print Footer

(Optional) Indicates whether to print a footer on an SAP report. When selected the agent prints a report with a footer.

JIL attribute: footer

Print Priority

(Optional) Specifies the priority of the SAP spool request. The priority is a number between 1 (highest priority) and 9 (lowest priority).

Limits: 0-9

JIL attribute: print_priority

Recipient Name

(Optional) Specifies the spool request recipient's name to appear on the SAP cover page, if applicable. This parameter corresponds to the SAPGUI Cover sheets options, Recipient field on the Background Print Parameters dialog.

Limits: Up to 16 characters; case-sensitive

Note: You can click  to open the Get SAP Office Users dialog to locate the spool request recipient name.

JIL attribute: recipient_name

Request Type

(Optional) Specifies the print request type.

Limits: Up to 64 characters; case-sensitive

JIL attribute: req_type

SAP Cover Page

(Optional) Indicates whether to include an SAP cover page containing information such as recipient name, department name, and format used, as follows:

- Y—Prints the SAP cover page
- N—Does not print the SAP cover page

JIL attribute: sap_banner

Spool Name

(Optional) Specifies the name of the print spool.

Limits: Up to 12 characters; case-sensitive

JIL attribute: spool_name

Spool Retention

(Optional) Specifies the number of days before the spool request is deleted.

Limits: 0-9

JIL attribute: expiration

Title

(Optional) Specifies the spool request description.

Limits: Up to 16 characters

JIL attribute: title

Define an SAP Event Monitor Job

You can define an SAP Event Monitor job to schedule workload based on the activity of an SAP event or to 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. Create an SAP Event Monitor (SAPEVT) job in either Quick Edit or Application Editor.

The Properties section for the SAP Event Monitor job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Event ID

Specifies the name of the SAP event to monitor or trigger.

Note: You can click  to open the Browse Background Event Identifiers dialog to locate an SAP event for the job definition.

3. (Optional) Specify optional [SAP Event Monitor properties](#) (see page 348).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The SAP Event Monitor job is defined.

More information:

[SAP Event Monitor Job Example](#) (see page 347)

SAP Event Monitor Job Example

The following examples describe sample SAP Event Monitor jobs.

Example: Trigger an SAP Event

This example defines an SAPEVT job with the Event trigger property selected, so the job triggers the SAP_TEST event by default.

To define an SAP Event Monitor job

1. Create an SAP Event Monitor job.
2. Enter the following properties:
 - Name—SAPEVT_job
 - Send to machine—sapagent
 - Owner—WAAESAP@sapagent
 - Logon Parameters: RFC destination—BI1
 - Event ID—SAP_TEST
 - Event trigger—selected
3. Commit the job.

Example: Monitor an SAP Event

This example defines an SAPEVT job without the Event trigger property selected, so the job monitors the SAP_TEST event and remains in RUNNING status.

To define an SAP Event Monitor job

1. Create an SAP Event Monitor job.
2. Enter the following properties:
 - Name—trigger_evt
 - Send to machine—localhost
 - Owner—WAAESAP@sapagent
 - Logon Parameters: Client—001
 - Logon Parameters: Language—EN
 - Logon Parameters: RFC destination—BI1
 - Event ID—SAP_TEST
 - Event parameters—L L
 - Event trigger—unselected
3. Commit the job.

SAP Event Monitor Properties

The SAP Event Monitor category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Continuous

(Optional) Indicates whether to monitor for the conditions continuously. When selected, each time the specified conditions occur, an alert is written to the scheduler log file.

Default: unselected (the job does not monitor for the conditions)

Note: To end continuous monitoring, you must complete the job manually.

JIL attribute: continuous

Event ID

Specifies the name of the SAP event to monitor or trigger.

Limits: Up to 32 valid SAP characters; case-sensitive

Note: You can click  to open the Browse Background Event Identifiers dialog to locate an SAP event.

JIL attribute: sap_event_id

Event parameter

(Optional) Specifies the name of the SAP event parameter.

Limits: Up to 64 characters

JIL attribute: sap_event_parm

Event trigger

(Optional) Indicates whether to trigger or monitor an SAP event. When selected, the agent triggers an SAP event.

Default: unselected (the job monitors for an SAP event)

JIL attribute: sap_is_trigger

Logon Parameters: Client

(Optional) Specifies the client within the SAP system. The value corresponds to the General data, Target server field on the Define Background Job dialog.

Limits: Up to three digits

JIL attribute: sap_client

Logon Parameters: Language

(Optional) Specifies a character code representing a valid language for SAP.

Limits: Up to two characters; case-sensitive

JIL attribute: sap_lang

Logon Parameters: RFC destination

(Optional) Indicates the destination specified in the connection properties file during installation.

Limits: Up to 256 characters; case-sensitive

JIL attribute: sap_rfc_dest

Define an SAP Job Copy Job

You can define an SAP Job Copy job to schedule 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. Create an SAP Job Copy (SAPJC) job in either Quick Edit or Application Editor.

The Properties section for the SAP Job Copy job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

SAP job name

Specifies the SAP R/3 job name.

SAP job count

Specifies the ID of the job to be copied.

3. (Optional) Specify optional [SAP Job Copy properties](#) (see page 351).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The SAP Job Copy job is defined.

More information:

[Windows Service Monitoring Job Examples](#) (see page 293)

SAP Job Copy Job Example

The following example describes a sample SAP Job Copy job.

Example: Define an SAP Job Copy Job

This example defines an SAP Job Copy job that copies the AM job with job count 11331500.

To define an SAP Job Copy job

1. Create an SAP Job Copy job.
2. Enter the following properties:
 - Name—SAPJC_job
 - Send to machine—sapagent
 - Owner—WAAESAP@sapagent
 - SAP job count—11331500
 - SAP job name—AM
3. Save the job.

SAP Job Copy Properties

The SAP Job Copy category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Failure message

(Optional) 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.

Limits: Up to 128 valid SAP characters; case-sensitive

JIL attribute: sap_fail_msg

Logon Parameters: Client

(Optional) Specifies the client within the SAP system. The value corresponds to the General data, Target server field on the Define Background Job dialog.

Limits: Up to three digits

JIL attribute: sap_client

Logon Parameters: Language

(Optional) Specifies a character code representing a valid language for SAP.

Limits: Up to two characters; case-sensitive

JIL attribute: sap_lang

Logon Parameters: RFC destination

(Optional) Indicates the destination specified in the connection properties file during installation.

Limits: Up to 256 characters; case-sensitive

JIL attribute: sap_rfc_dest

Monitor children

(Optional) Indicates whether to monitor a job's children. Children jobs are jobs produced by a parent job. When selected, the agent monitors the job's children.

JIL attribute: sap_mon_child

Release option

(Defaulted) Specifies the action to take with an SAP R/3 job after it is defined, as follows:

- Immediately—Releases the job immediately
- As soon as possible—Releases the job as soon as possible

Default: As soon as possible

JIL attribute: sap_release_option

SAP job count

Specifies the ID of the job to be copied.

Limits: Up to 8 digits

JIL attribute: sap_job_count

SAP job name

Specifies the SAP R/3 job name.

Limits: Up to 32 characters; case-sensitive

Note: You can click  to open the Search SAP Job Name and Job Count dialog to locate an SAP job.

JIL attribute: sap_job_name

Step number

(Optional) Specifies the number of the first step to start copying job data from.

Limits: 0-*n*, where *n* is the highest step number

Note: If you specify 0 or 1, all steps are copied.

JIL attribute: sap_step_num

Success message

(Optional) Specifies a string that, when found in the condition output of an SAP job, indicates the success of the job.

Limits: Up to 128 characters; case-sensitive

JIL attribute: sap_success_msg

Target job name

(Optional) Specifies the name of the target job to copy. If unspecified, the target job will have the same name as the source job.

Limits: Up to 32 characters

JIL attribute: sap_target_jobname

Target system

(Optional) Specifies the host computer within the SAP system architecture that is capable of running SAP system workload.

Limits: Up to 256 characters; case-sensitive

Note: You can click  to open the Browse SAP Target Systems dialog to locate the host computer.

JIL attribute: sap_target_sys

Define an SAP Process Monitor Job

You can define an SAP Process Monitor 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. Create an SAP Process Monitor (SAPPM) job in either Quick Edit or Application Editor.

The Properties section for the SAP Process Monitor job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Process status

Specifies the SAP process status to monitor, as follows:

- Waiting
- Running
- Stopped

3. (Optional) Specify optional [SAP Process Monitor properties](#) (see page 357).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The SAP Process Monitor job is defined.

More information:

[SAP Process Monitor Job Example](#) (see page 355)

SAP Process Monitor Job Example

The following example describes a sample SAP Process Monitor job.

Example: Define an SAP Process Monitor Job

This example defines an SAP Process Monitor job that monitors the RUNNING status.

To define an SAP Process Monitor job

1. Create an SAP Process Monitor job.
2. Enter the following properties:
 - Name—test_SAPPMM
 - Send to machine—sapagent
 - Logon Parameters: RFC destination—BI1
 - Process status—Running
3. Commit the job.

Using Success and Failure Messages within an SAP Job

You can check the output of an SAP R/3 job for specific text strings to determine whether the job is a success or a failure. You specify the text string in the Success message and Failure message SAP R/3 properties in the job definition. For example, suppose that when you cancel an SAP R/3 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 either the Success message property or Failure message property within the Step parameter description on the Step parameters - Command tab.

For more flexibility, you can specify regular expressions instead of simple text strings within the Success message and Failure message properties. 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 an Internet search for java pattern.

Note: To enable regular expression processing, the agent administrator must configure the agent, setting the `sap.useRegularExpressions` parameter to true.

Examples: Using Regular Expressions

- The following expression checks for "TEST" in the job's output file:
`(?s).*TEST.*`
Note: To check multiple lines in the file, you must use `(?s)`. The first `.`* indicates that any number of characters can precede TEST. The second `.`* indicates that any number of characters can follow it.
- The following expression checks whether "not found" or "started" appears in the job's output file:
`<(?s).*[not found|started].*>`
- The following expression checks whether "Job canceled" appears in the job's output file:
`(?s).*Job\scanceled.*`

More information:

[SAP R/3 Properties](#) (see page 363)

[Step Parameters - Command Properties](#) (see page 372)

SAP Process Monitor Properties

The SAP Process Monitor category includes the following properties:

ABAP name

(Optional) Starts a step definition within the job definition and identifies the ABAP step to be run. This property corresponds to the SAPGUI ABAP program Name field on the Create Step dialog.

Limits: Up to 80 characters

Note: This property is not valid if the process status is waiting.

JIL attribute: sap_abap_name

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Continuous

(Optional) Indicates whether to monitor for the conditions continuously. When selected, each time the specified conditions occur, an alert is written to the scheduler log file.

Default: unselected (the job does not monitor for the conditions)

Note: To end continuous monitoring, you must complete the job manually.

JIL attribute: continuous

Logon Parameters: Client

(Optional) Specifies the client within the SAP system. The value corresponds to the General data, Target server field on the Define Background Job dialog.

Limits: Up to three digits

JIL attribute: sap_client

Logon Parameters: Language

(Optional) Specifies a character code representing a valid language for SAP.

Limits: Up to two characters; case-sensitive

JIL attribute: sap_lang

Logon Parameters: RFC destination

(Optional) Indicates the destination specified in the connection properties file during installation.

Limits: Up to 256 characters; case-sensitive

JIL attribute: sap_rfc_dest

Process client

(Optional) Specifies the client monitored for the process.

Limits: Up to three digits

Note: This property is not valid if the process status is waiting.

JIL attribute: sap_process_client

Process status

Specifies the SAP process status to monitor.

- Running
- Stopped
- Waiting

JIL attribute: sap_process_status

Process type

(Optional) Specifies the SAP business process type to monitor.

- BGD—background
- DIA—dialog
- ENQ—lock
- SPO—spool
- UPD—update
- UP2—update2

JIL attribute: sap_proc_type

Process user

(Optional) Specifies an SAP user name. When a process matches the specified process status, the job the process runs is checked for a match to this user.

Limits: Up to 80 characters; case-sensitive

Note: This property is not valid if the process status is waiting.

JIL attribute: sap_proc_user

Target system

(Optional) Specifies the host computer within the SAP system architecture that is capable of running SAP system workload.

Limits: Up to 256 characters; case-sensitive

Note: You can click  to open the Browse SAP Target Systems dialog to locate the host computer.

JIL attribute: sap_target_sys

Define an SAP R/3 Job

You can define an SAP R/3 job to schedule an SAP R/3 job on your SAP system. You use this procedure to create a new job. You can also copy a job from the SAP system to save time.

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

To schedule an SAP r/3 job, you must define at least one ABAP (SAP program) in the job definition.

To define an SAP R/3 job

1. Create an SAP R/3 (SAP) job in either Quick Edit or Application Editor.

The Properties section for the SAP R/3 job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

SAP job name

Specifies the SAP R/3 job name.

Step parameters

Specifies the [SAP R/3 step specifications](#) (see page 372).

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.

3. (Optional) Specify optional [SAP R/3 properties](#) (see page 357).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The SAP R/3 job is defined.

More information:

[Copy a Job from the SAP System to a Job Flow](#) (see page 323)
[Using Success and Failure Messages within an SAP Job](#) (see page 356)
[Define Recipients for the Job's Output](#) (see page 367)
[Define Print or Archive Parameters for a Step](#) (see page 370)
[Send the SAP Spool File by Email on Step Completion or Failure](#) (see page 380)
[SAP R/3 Job Examples](#) (see page 361)

SAP R/3 Job Examples

The following examples describe sample SAP R/3 jobs.

Example: Define an SAP R/3 Job with One Step

This example runs the SAP R/3 job named BI_WRITE_PROT_TO_APPLLOG, which in turn runs the SAP program (ABAP) named RSBATCH_WRITE_PROT_TO_APPLLOG.

To define an SAP R/3 job with one step

1. Create an SAP R/3 job.
2. Enter the following properties:
 - Name—test_SAP
 - Send to machine—sapagent
 - Owner—WAAESAP@sapagent
 - SAP job name—BI_WRITE_PROT_TO_APPLLOG
 - Logon Parameters: RFC destination—BI1
3. Open the Step Parameters dialog.
4. Enter the following properties on the Command tab:
 - ABAP Name—RSBATCH_WRITE_PROT_TO_APPLLOG
 - ABAP Language—EN
5. Click OK.

The step specifications are added to the job definition.
6. Commit the job.

Example: Define an SAP R/3 Job with Multiple Steps

This example runs an SAP R/3 job named SAP_2_STEPS, which in turn runs two steps. Each step runs the same ABAP but with different parameters.

To define an SAP R/3 job with multiple steps

1. Create an SAP R/3 job.
2. Enter the following properties:
 - Name—test_1
 - Send to machine—sapagent
 - Owner—WAAESAP@sapagent
 - SAP job name—SAP_2_STEPS
 - Release option—Immediately
3. Open the Step Parameters dialog.
4. Enter the following properties on the Command tab:
 - ABAP Name—BTCTEST
 - Variant—test
 - ABAP Language—EN
5. Enter the following properties on the Archive tab:
 - Printer—LP01
 - ArchiveLink Object Type—ARCHIVE
 - Document Type—ARCHIVE
 - ArchiveLink Information—inf,num
6. Enter the following properties on the Output tab:
 - Archiving Mode—archived
 - Number of Copies—2
 - Number of Columns—80
 - Number of Lines—65

7. Click OK.
8. Open the Step Parameters dialog for the next step.
9. Enter the following properties on the Command tab:
 - ABAP Name—BTCTEST
 - Variant—test
 - ABAP Language—EN
10. Enter the following properties on the Archive tab:
 - Printer—LP01
 - ArchiveLink Object Type—ARCHIVE
 - Document type—ARCHIVE
 - ArchiveLink Information—inf,num
11. Enter the following properties on the Output tab:
 - Archiving Mode—archived
 - Number of Copies—1
 - Number of Columns—80
 - Number of Lines—65
 - SAP Cover Page—Yes
 - Print Cover Page—selected
 - Recipient Name—cyber
12. Click OK.
13. Commit the job.

SAP R/3 Properties

The SAP R/3 category includes the following properties:

Agent job class

Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Failure message

(Optional) Specifies a string that, when found in the condition output of an SAP job, indicates the failure of the job.

Limits: Up to 128 valid SAP characters; case-sensitive

JIL attribute: sap_fail_msg

Logon Parameters: Client

(Optional) Specifies the client within the SAP system. The value corresponds to the General data, Target server field on the Define Background Job dialog.

Limits: Up to three digits

JIL attribute: sap_client

Logon Parameters: Language

(Optional) Specifies a character code representing a valid language for SAP.

Limits: Up to two characters; case-sensitive

JIL attribute: sap_lang

Logon Parameters: RFC destination

(Optional) Indicates the destination specified in the connection properties file during installation.

Limits: Up to 256 characters; case-sensitive

JIL attribute: sap_rfc_dest

Monitor children

(Optional) Indicates whether to monitor a job's children. Children jobs are jobs spawned by a parent job. When selected, the agent monitors the job's children.

JIL attribute: sap_mon_child

Office output

(Optional) Indicates whether to save outgoing documents to the SAPoffice outbox of the SAP user associated with the job. When selected, the agent saves outgoing documents.

JIL attribute: sap_office

Release option

(Defaulted) Specifies the action to take with a job after it is defined, as follows:

- Immediately—Releases the job immediately
- As soon as possible—Releases the job as soon as possible

Default: As soon as possible

JIL attribute: sap_release_option

SAP job class

(Optional) Specifies a valid SAP system job class.

Limits: One character

JIL attribute: sap_job_class

SAP job name

(Optional) Specifies the SAP R/3 job name.

Limits: Up to 32 characters; case-sensitive

JIL attribute: sap_job_name

Spool list recipients

(Optional) Specifies the [recipient properties](#) (see page 369) for the job's output (spool file).

Limits: Up to 4096 characters

JIL attribute: sap_recipients

Step parameters

Specifies the [SAP R/3 step specifications](#) (see page 372).

Limits: Up to 4096 characters

Note: To add step parameters, select Add and click Go. A row appears below the Step parameter field. Click  to open the Step Parameters dialog to specify the parameters for the job definition.

JIL attribute: sap_step_parms

Success message

(Optional) Specifies a string that, when found in the condition output of an SAP job, indicates the success of the job.

Limits: Up to 128 characters; case-sensitive

JIL attribute: sap_success_msg

Target system

(Optional) Specifies the host computer within the SAP system architecture that is capable of running SAP system workload.

Limits: Up to 256 characters; case-sensitive

Note: You can click  to open the Browse SAP Target Systems dialog to locate the host computer.

JIL attribute: sap_target_sys

Define Recipients for the Job's Output

In an SAP R/3 or a Batch Input Session job, you can define recipients for the job's output (spool file). The recipient can be an email address, an SAPoffice distribution list, or an SAPoffice user.

To define recipients for the job's output

1. Open the SAP R/3 or Batch Input Session job for which you want to define recipients for the job's output.
2. Select Add and click Go in the Spool list recipients field.
A row appears below the Spool list recipients field.
3. Click  beside the field.
The Spool List Recipient dialog opens.
4. Select a recipient type, as follows:
 - INT—Specifies an email address
 - DLI—Specifies an SAPoffice distribution list
 - SU—Specifies an SAP user
5. Select whether to send the job's output using the To, cc, or bcc option in the Recipient field.
6. Specify the recipient name in the Recipient field.
7. (Optional) Select one or more of the following SAPoffice options to control delivery of the job output:
 - No forward—Lets SAPoffice users forward the document.
 - No print—Lets SAPoffice users print the document.
 - Express—Sends instant messages to recipients if they are logged in to SAP.

Note: These options apply to the DLI and SU recipient types only.
8. Click OK.
The spool list recipients are added to the job definition.
9. Commit the job.

Example: Send a Spool File to a Distribution List

This example sends the spool file to the users on the payroll distribution list.

To send a spool file to a distribution list

1. Open the Spool List Recipient dialog.
2. Edit the following properties:
 - Recipient Type—DLI
 - Recipient—To, payroll
3. Click OK.

Example: Send a Carbon Copy of a Spool File

This example sends a carbon copy of a job spool file to the email address jsmith@company1.com.

To send a carbon copy of a spool file

1. Open the Spool List Recipient dialog.
2. Enter the following properties:
 - Recipient Type—INT
 - Recipient—cc, jsmith@company1.com
3. Click OK.

Example: Save an Outgoing Document to an SAPoffice Outbox

This example saves outgoing documents to the SAPoffice outbox of the SAP user J01Prod.

To save an outgoing document to an SAPoffice outbox

1. Open the Spool List Recipient dialog.
2. Enter the following properties:
 - Recipient Type—SU
 - Recipient—To, J01PROD
3. Select the SAPoffice option.
4. Click OK.

More information:

[Spool List Recipient Properties](#) (see page 369)

Spool List Recipient Properties

The Spool List Recipient dialog includes the following properties:

Recipient Type

Specifies the recipient type, as follows:

- DLI—SAPoffice distribution list
- INT—Email address (requires the mailto keyword)
- SU—SAP user

JIL attribute: rcp_type

Recipient

Specifies the target recipient's email address. This property is required if the spool list recipient exists.

Limits: Up to 256 characters

JIL attribute: recipient

SAPOffice Options

(Optional) Specifies recipient options when the Recipient type is DLI or SU. The following options are available:

- Express—Sends instant messages to recipients if they are logged in to SAP.

JIL attribute: express

- No forward—Lets SAPoffice users forward the document.

JIL attribute: no_forward

- No print—Lets SAPoffice users print the document.

JIL attribute: no_print

Define Print or Archive Parameters for a Step

In an SAP R/3 or a Batch Input Session job, you can define print or archive parameters, or both, for a step.

To define print or archive parameters for a step

1. Open the SAP R/3 or Batch Input Session job for which you want to define print or archive parameters.
2. Select Add and click Go in the Step parameters field.
A row appears below the Spool list recipients field.
3. Click  beside the field.
The Step Parameters dialog opens.
4. Specify the following required property on the Command tab:

ABAP Name

Defines the valid SAP system ABAP name.

5. (Optional) Specify optional [command parameters](#) (see page 372) for the step.
6. Click the Output tab.
7. Specify the following required properties on the Output tab.

Output Device

Specifies the name of the output device.

Archiving Mode

Specifies whether the spool file is printed, archived, or both.

8. (Optional) Specify optional output parameters for the step.
9. Click the Archive tab.
10. (Optional) Specify optional [archive parameters](#) (see page 377) for the step.
11. Click OK.

The parameters for the step are added to the job definition.

Example: Specify Archive Parameters

In this example, the archive target printer is PRT1. The archive mode, document type, and external archive object are Archive. The archive link information is inf and the archive link client is 800.

To specify archive parameters

1. Open the Step Parameters dialog for the step.
2. Enter the following property on the Output tab:
 - Archive Mode—Archive
3. Enter the following properties on the Archive tab:
 - ArchiveLink Object Type—ARCHIVE
 - Document Type—ARCHIVE
 - ArchiveLink Information—info
 - Printer—PRT1
 - Client—800
4. Click OK.

Example: Specify Print Parameters

In this example, the print destination for the RSPARAM ABAP is LP01. The number of lines per list page is 65 and the line width of the list is 80. One copy of the report associated with the ABAP is printed.

To specify print parameters

1. Open the Step Parameters dialog for the step.
2. Enter the following property on the Command tab:
 - ABAP Name—RSPARAM
3. Enter the following properties on the Output tab:
 - Archiving Mode—archived
 - Number of Copies—1
 - Number of Columns—80
 - Number of Lines—65
4. Enter the following properties on the Archive tab:
 - ArchiveLink Object Type—Archive
 - Document Type—Archive
 - ArchiveLink Information—info
 - Printer—LP01
 - Client—800
5. Click OK.

Step Parameters - Command Properties

The Step Parameters dialog includes the following properties on the Command tab, which define the ABAP (step):

ABAP Language

(Optional) Specifies a language for the ABAP.

Limits: Up to 16 characters

Note: The SAP system default logon language is the default.

JIL attribute: abap_lang

ABAP Name

Defines the valid SAP system ABAP name. The ABAP name starts a step definition within the job definition and identifies the ABAP step to be run.

Limits: Up to 40 characters

Note: This parameter corresponds to the SAPGUI ABAP program Name field on the Create Step dialog.

JIL attribute: abap_name

Failure Message

(Optional) 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.

Limits: Up to 128 valid SAP characters; case-sensitive

JIL attribute: sap_fail_msg

Step User

(Optional) Specifies the SAP R/3 system user under whose authorization the ABAP program runs.

Limits: Up to 16 characters

JIL attribute: step_user

Success Message

(Optional) 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.

Limits: Up to 128 valid SAP characters; case-sensitive

JIL attribute: sap_success_msg

Variant

(Optional) Specifies the variant to pass.

Limits: Up to 14 characters

JIL attribute: variant

Step Parameters - Output Properties

The Step Parameters dialog includes the following properties on the Output tab:

Archiving Mode

(Optional) Specifies whether the spool file is printed, archived, or both.

JIL attribute: prt_arc_mode

Authorization

(Optional) Specifies the password for viewing the print spool list.

Limits: Up to 12 characters; case-sensitive

JIL attribute: authorization

Dataset Name

(Optional) Specifies the print spool data set name.

Limits: Up to six characters; case-sensitive

JIL attribute: dataset

Delete After Output

(Optional) Indicates whether to delete the print spool file after output. When selected, the agent deletes the print spool file.

JIL attribute: release

Department

(Optional) Specifies the department name to print on the cover page.

Limits: Up to 12 characters; case-sensitive

JIL attribute: dept

Format

(Optional) Specifies an output format for an SAP report.

Limits: Up to 64 characters; case-sensitive

JIL attribute: format

New Spool Request

(Optional) Indicates whether to create a new spool request. When selected, the agent creates a new spool request. SAP does not append the spool request to an existing spool request.

JIL attribute: new_spool

Number of Columns

(Optional) Specifies the number of columns for the SAP report.

Limits: Up to 10 characters

JIL attribute: num_columns

Number of Copies

(Optional) Specifies the number of copies to print of the SAP report.

Limits: Up to 10 digits

JIL attribute: copies

Number of Lines

(Optional) Specifies the number of lines for the SAP report.

Limits: Up to 10 digits

JIL attribute: num_lines

OS Cover Page

(Optional) Specifies whether to print the operating system page, as follows:

- No
- Yes
- Default

JIL attribute: host_page

Output Device

Specifies the name of the output device.

Limits: Up to 16 characters

JIL attribute: dest

Print Cover Page

(Optional) Indicates whether to print the selection cover page. When selected, the agent prints the selection cover page.

JIL attribute: banner_page

Print Footer

(Optional) Indicates whether to print a footer on the SAP report. When selected, the agent prints a footer.

JIL attribute: footer

Print Immediately

(Optional) Indicates whether to print the job immediately after completion. When selected, the agent prints the job immediately.

JIL attribute: print_imm

Print Priority

(Optional) Specifies the priority number of the print spool request.

Limits: 0-9

JIL attribute: print_priority

Recipient Name

(Optional) Specifies the recipient on the SAP cover page.

Limits: Up to 16 characters

JIL attribute: recipient

Request Type

(Optional) Specifies the print request type.

Limits: Up to 64 characters

JIL attribute: req_type

SAP Cover Page

(Optional) Specifies whether to print the SAP cover page, as follows:

- No
- Yes
- Default

JIL attribute: sap_banner

Spool Name

(Optional) Specifies the spool request name.

Limits: Up to 12 characters

JIL attribute: spool_name

Spool Retention

(Optional) Specifies the number of days before the spool request expires and is deleted.

Limits: 0-9

JIL attribute: expiration

Title

(Optional) Specifies the spool request description.

Limits: Up to 16 characters

JIL attribute: title

Step parameters - Archive Properties

The Step Parameters dialog includes the following properties on the Archive tab:

ArchiveLink Host

(Optional) Specifies the SAP ArchiveLink RPC host.

Limits: Up to 32 characters

JIL attribute: arc_host_link

ArchiveLink Information

(Optional) Specifies the SAP ArchiveLink RPC information.

Limits: Up to 3 characters

JIL attribute: arc_info

ArchiveLink Object Type

(Optional) Specifies the SAP ArchiveLink object type.

Limits: Up to 10 characters

JIL attribute: arc_obj_type

Archiving Path

(Optional) Specifies the standard archive path.

Limits: Up to 70 characters; case-sensitive

JIL attribute: arc_path

Client

(Optional) Specifies the SAP ArchiveLink client.

Limits: Up to 10 characters

JIL attribute: arc_client

Connection Name

(Optional) Specifies the SAP ArchiveLink communication connection.

Limits: Up to 14 characters; case-sensitive

JIL attribute: arc_connect

Date

(Optional) Specifies the date to do the archiving in YYYYMMDD format.

Limits: Up to 8 characters

JIL attribute: arc_date

Document Class

(Optional) Specifies the SAP ArchiveLink document class.

Limits: Up to 20 characters

JIL attribute: arc_doc_class

Document Type

(Optional) Specifies the SAP ArchiveLink document type.

Limits: Up to 10 characters

JIL attribute: arc_doc_type

Format

(Optional) Specifies the SAP ArchiveLink output format.

Limits: Up to 16 characters; case-sensitive

JIL attribute: arc_format

Printer

(Optional) Specifies the SAP ArchiveLink target printer.

Limits: Up to 4 characters; case-sensitive

JIL attribute: arc_printer

Protocol

(Optional) Specifies the archive storage connection protocol.

Limits: Up to 8 characters; case-sensitive

JIL attribute: arc_protocol

Report

(Optional) Specifies the SAP ArchiveLink report name.

Limits: Up to 30 characters; case-sensitive

JIL attribute: arc_report

Service Destination

(Optional) Specifies the SAP ArchiveLink RPC service/destination.

Limits: Up to 32 characters; case-sensitive

JIL attribute: arc_service

Target Storage System

(Optional) Specifies the SAP ArchiveLink target storage system.

Limits: Up to 2 characters

JIL attribute: arc_storage

Text

(Optional) Specifies the archiving text.

Limits: Up to 30 characters; case-sensitive

JIL attribute: arc_text

User

(Optional) Specifies the user ID name for archiving the report.

Limits: Up to 16 characters

JIL attribute: arc_userid

Version

(Optional) Specifies the archive version.

Limits: Up to 4 characters

JIL attribute: arc_version

Send the SAP Spool File by Email on Step Completion or Failure

In an SAP R/3 or a Batch Input Session job, you can send the SAP spool file by email on step completion or failure.

To send the SAP spool file by email on step completion or failure

1. Specify the ABAPNAME statement to start the step definition within the job definition.
2. Specify the EMAILADDR statement after the ABAPNAME statement.
To specify multiple email addresses, separate each address with a comma and enclose the entire list of addresses in single quotation marks.
3. (Optional) Specify the following optional statements to determine whether the step is a success or failure based on a text string in the job's output:
 - SUCCESSMSG
 - FAILUREMSG

Note: This step does not apply to Batch Input Session jobs.

Example: Email Spool File to Two Addresses

This example emails the spool file to jsmith and djones upon completion or failure of the BTCTEST ABAP.

```
AGENT SAPHTAGENT
SAPUSER J01PROD
ABAPNAME BTCTEST
VARIANT TEST
EMAILADDR 'jsmith@company1.com,djones@company1.com'
```

Chapter 17: PeopleSoft Jobs

This section contains the following topics:

[PeopleSoft Jobs](#) (see page 381)

[Define a PeopleSoft Job](#) (see page 382)

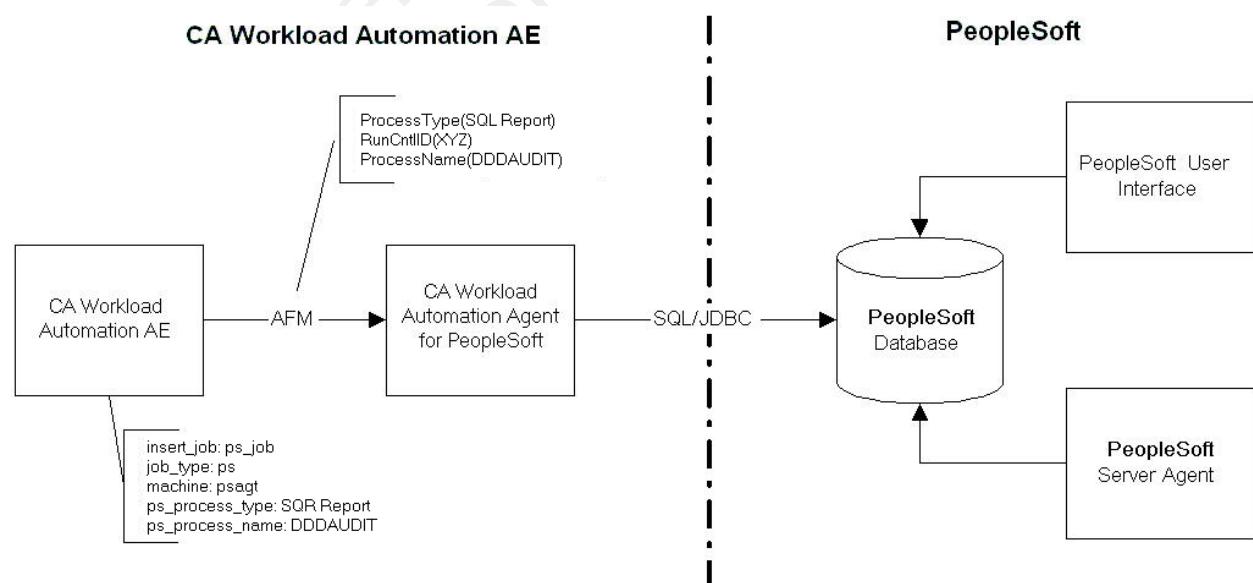
PeopleSoft Jobs

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 PSPRCRSRQST table in the PS database. The following diagram shows the functional relationship between the scheduling manager, the agent, and the PeopleSoft system:



Define a PeopleSoft Job

You can define a PeopleSoft 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. Create a PeopleSoft (PS) job in either Quick Edit or Application Editor.
The Properties section for the PeopleSoft job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Process name

Specifies the name of the PeopleSoft process to run. This value corresponds to the Process Name field in PeopleSoft.

Process type

Specifies the PeopleSoft process type corresponding to the process that you want to run. PeopleSoft stores the list of process types in the PS_PRCSTYPEDEFN table. This value corresponds to the Process Type field in PeopleSoft.

Run control ID

Specifies the value assigned to the run control identifier. This value corresponds to the Run Control ID field in PeopleSoft.

3. (Optional) Specify optional [PeopleSoft properties](#) (see page 388).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The PeopleSoft job is defined.

More information:

- [Mapping of PeopleSoft Field Names to PeopleSoft Job Properties](#) (see page 383)
[PeopleSoft Properties](#) (see page 388)
[PeopleSoft Job Examples](#) (see page 384)

Mapping of PeopleSoft Field Names to PeopleSoft Job Properties

When you define a PeopleSoft job, you specify PeopleSoft job properties that map to your PeopleSoft process request. The following table maps the PeopleSoft fields to the properties:

| PeopleSoft Field Name | PeopleSoft Job Property |
|-------------------------|---------------------------------------|
| Format | Output format |
| Type | Output type |
| Folder Name | Folder name |
| ID Type (Role selected) | Roles distribution list |
| Distribution ID | |
| ID Type (User selected) | User distribution list |
| Distribution ID | |
| Email Address List | Email address, Expanded email address |
| Email With Log | Email with log |
| Email Subject | Email subject |
| Message Text | Message text |
| Email Web Report | Email web report |
| Output Destination | Output destination path |
| Process Name | Process name |
| Process Type | Process type |
| Run Control ID | Run control ID |
| Server Name | Server name |
| Time Zone | Time zone |

PeopleSoft Job Examples

The following examples describe sample PeopleSoft jobs.

Example: Run a PeopleSoft Process

This example runs an Application Engine process named DDDAUDIT. The job runs on the agent named psagt.

To run a PeopleSoft process

1. Create a PeopleSoft job.
2. Enter the following properties:
 - Name—audit
 - Send to machine—psagt
 - Process name—DDDAUDIT
 - Process type—Application Engine
3. Commit the job.

Note: The job uses the default 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.

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.

To distribute a report to users

1. Create a PeopleSoft job.
2. Enter the following properties:
 - Name—ps_users
 - Send to machine—psagt
 - Process name—XRFWIN
 - Process type—Crystal
 - Run control ID—test
 - Output destination type—Email
 - Output destination format—PDF
 - User distribution list—VP1,VP2,VP3
3. Commit the job.

Example: Email a PeopleSoft Report

This example runs a Crystal report and emails the output to recipients. The Crystal report runs under the VP3 operator ID. The output is sent to the email addresses specified in the ps_email_address attribute. The email includes a subject title and body text.

To email a PeopleSoft report

1. Create a PeopleSoft job.
2. Enter VP3@ps1in the Owner field.
3. Enter the following properties:
 - Name—ps_email
 - Send to machine—psagt
 - Process name—XRFWIN
 - Process type—Crystal
 - Run control ID—test
 - Operator ID—VP3@ps1
 - Output destination type—Email
 - Output destination format—PDF
 - Email subject—PeopleSoft Report Status
 - Message text—This report is available for distribution
 - Email address—user1@example.com;user2@example.com
4. Commit the job.

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.

To format a PeopleSoft job output as an HTML web report

1. Create a PeopleSoft job.
2. Enter VP3@ps1 in the Owner field.
3. Enter the following properties:
 - Name—ps_htmfile
 - Send to machine—psagt
 - Process name—XRFWIN
 - Process type—SQR Report
 - Operator ID—VP3@ps1
 - Server name—PSPR
 - Output destination type—Web
 - Output destination format—HTM
 - Run control ID—PS_ALL
4. Commit the job.

Example: Send a Job's Output to a Specified Printer

This example runs an SQR Report. The report is formatted as PS and sent as output to a printer.

To send a job's output to a specified printer

1. Create a PeopleSoft job.
2. Enter VP3@ps1in the Owner field.
3. Enter the following properties:
 - Name—ps_printer
 - Send to machine—psagt
 - Process name—XRFWIN
 - Process type—SQR Report
 - Operator ID—VP3@ps1
 - Output destination type—PRINTER
 - Output destination format—PS
 - Output destination path—\\printers\PRINTER1
 - Run control ID—test
4. Commit the job.

PeopleSoft Properties

The PeopleSoft category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Arguments

(Optional) Specifies additional arguments to append to the parameter list in the PeopleSoft database in the PSPRCSPARMS.PARMLIST field.

Limits: Up to 254 characters

JIL attribute: ps_args

Disable restart

(Optional) Indicates whether to disable a restart feature for previously-failed jobs. When selected, previously-failed jobs restart from the beginning. When unselected, a failed job that is resubmitted restarts from where it stopped.

Note: This option is used primarily in Application Engine programs.

JIL attribute: ps_restarts

Distribution: Email address

Specifies the email addresses of the recipients on a distribution list. The PeopleSoft report is emailed to the recipients. You can specify multiple email addresses.

Limits: Up to 256 characters; separate each address with a semi-colon

Note: This property applies only if you specify Email or Web in the Output type field.

JIL attribute: ps_email_address

Distribution: Email subject

(Optional) Defines an email subject to include in the email to distribute to recipients of a PeopleSoft report. This value corresponds to the Email Subject field in PeopleSoft.

Limits: Up to 256 characters

Note: This property applies only if you specify Email or Web in the Output type field.

JIL attribute: ps_email_subject

Distribution: Expanded email address

(Optional) Specifies additional email addresses.

Limits: Up to 256 characters

Note: This property applies only if you specify Email or Web in the Output type field.

JIL attribute: ps_email_address_expanded

Distribution: Folder name

(Optional) Specifies the name of a distribution detail folder. This value corresponds to the Folder Name field in PeopleSoft.

Limits: Up to 18 characters

Note: This property applies only if you specify Email or Web in the Output type field.

JIL attribute: ps_detail_folder

Distribution: Message text

(Optional) Defines the body text of the email. This value corresponds to the Message Text field in PeopleSoft.

Limits: 1000 characters

Note: This property applies only if you specify Email or Web in the Output type field.

JIL attribute: ps_email_text

Distribution: Roles distribution list

(Optional) Specifies a role or a list of roles to send the report to. This value corresponds to the ID Type field (with Role selected) and the Distribution ID field in PeopleSoft.

Limits: Up to 256 characters; separate multiple roles with a comma

Example: Role1, Role2

Note: This property applies only if you specify Email or Web in the Output type field.

JIL attribute: ps_dlist_roles

Distribution: User distribution list

(Optional) Specifies a distribution list of operator IDs to send a PeopleSoft report to.

Limits: Up to 256 characters; separate multiple users with a comma

Example: User1, User2

Note: This property applies only if you specify Email or Web in the Output type field.

JIL attribute: ps_dlist_users

Distribution: Email web report

(Optional) Indicates whether to email a web report to recipients on a distribution list. When selected, the agent emails the web report.

JIL attribute: ps_email_web_report

Distribution: Email with log

(Optional) Indicates whether to email job logs along with the PeopleSoft report to recipients on a distribution list. When selected, the agent emails the job logs.

Default: unselected (the job does not email job logs)

JIL attribute: ps_email_log

Operator ID

(Optional) Specifies the PeopleSoft operator type that the operator ID belongs to.

Limits: Up to 5 characters

JIL attribute: ps_operator_id

Output destination path

(Optional) Specifies the output destination for the PeopleSoft request. The destination can be a file directory or a printer using the following format:

file_path | *printer_path*

file_path

Specifies the path to the output directory and the output file name.

Note: Specify this path when the output type is File.

Limits: Up to 127 characters; case-sensitive

Default: The PeopleSoft log/output directory

printer_path

Specifies the network location of the printer including the printer server and shared printer name.

Notes:

- Specify this path when the output type is Printer.
- The specified printer must be set up on the PeopleSoft system.
- Your agent administrator can specify a default printer by setting the ps.default.printer parameter in the agent's agentparm.txt file.
- The value specified for printer_path overrides the default printer specified in the agent's agentparm.txt.

Limits: Up to 127 characters; case-sensitive

JIL attribute: ps_output_dest

Output destination format

(Optional) Specifies the type of format for the report output, as follows:

- ANY—Any
- NONE—(None)
- PDF—Acrobat (*.pdf)
- CSV—Comma delimited (*.csv)
- HP—HP Format (*.lis)
- HTM—HTML Documents (*.htm)
- LP—Line printer format (*.lis)
- WKS—Lotus 1-2-3 files
- XLS—Microsoft Excel Files (*.xls)
- DOC—Microsoft Word (*.doc)
- PS—Postscript (*.lis)
- RPT—Crystal Report (*.rpt)
- RTF—Rich Text File (*.rtf)
- SPF—SQR Portable Format (*.spf)
- TXT—Text Files (*.txt)
- OTHER—Other
- Default—Default
- XML—XML Format (*.xml)
- DAT—Data Mover Data File (*.dat)

JIL attribute: ps_dest_format

Output destination type

(Optional) Specifies the output destination type for a PeopleSoft report, as follows:

- NONE—Specifies no output destination type.
- FILE—Sends the output of the PeopleSoft report to a file.
- PRINTER—Sends the output of the PeopleSoft report to a printer.
- EMAIL—Sends the output of the PeopleSoft report as an email message.
- WEB—Posts the output of the PeopleSoft report on a website.

JIL attribute: ps_dest_type

Process name

Specifies the name of the PeopleSoft process to run. This value corresponds to the Process Name field in PeopleSoft. PeopleSoft stores the list of process names in the PS_PRCDEFN table.

Limits: Up to 12 characters

JIL attribute: ps_process_name

Process type

Specifies the PeopleSoft process type corresponding to the process that you want to run. PeopleSoft stores the list of process types in the PS_PRCSTYPEREFN table. This value corresponds to the Process Type field in PeopleSoft.

The following types are available:

- Application Engine
- COBOL SQL
- Crw Online
- Crystal
- Crystal Check
- Cube Builder
- Database Agent
- Demand Planning Upload
- Essbase
- HyperCube Builder
- Message Agent API
- nVision
- nVision-Report
- Optimization Engine
- SQR PO-Special Process
- SQR Process
- SQR Report
- SQR Report For WF Delivery
- Winword

Default: Application Engine

Limits: Up to 30 characters; case-sensitive

JIL attribute: ps_process_type

Run control arguments

(Optional) Specifies the run control arguments for a run control table on the PeopleSoft system. You can specify multiple arguments.

Limits: Up to 1024 characters for each argument; separate each argument with a comma

Notes:

- You must also specify the run control table using the Run control table name property.
- The arguments must match the structure in the run control table.

JIL attribute: ps_run_cntrl_args

Run control ID

(Optional) Specifies the value assigned to the run control identifier. This value corresponds to the Run Control ID field in PeopleSoft.

Limits: Up to 30 characters; case-sensitive

Example: FLOOR8_COLOR

Note: In PeopleSoft, some processes spawn children using the CreateProcessRequest PeopleCode function. PeopleSoft may mark the parent process as a successful completion even if one or more of its children fail. If you want the agent to check the status of children when determining the completion status of the parent, add ".*" to the end of the Run Control ID, for example, PS_ALL.*.

JIL attribute: ps_run_cntrl_id

Run control table name

(Optional) Specifies the name of the table that contains the run parameters for a given PeopleSoft process.

Limits: Up to 50 characters

Note: You must specify run control arguments using the Run control arguments property.

JIL attribute: ps_run_control_table

Server name

(Optional) Specifies the name of the target server that runs the PeopleSoft job. PeopleSoft stores the list of server names in the PS_SERVERDEFN table. This value corresponds to the Server Name field in PeopleSoft.

Limits: Up to 8 characters; cannot contain delimiters (such as spaces)

Example: PSNT

JIL attribute: ps_server_name

Skip parameter updates

(Optional) Indicates whether you want the agent to update job parameters with data in the PS_PRCDEFN table. When selected, the agent does not update job parameters. When the updates are skipped, you can use the Run control arguments field to pass additional argument values.

JIL attribute: ps_skip_parm_updates

Time zone

(Optional) Specifies a different time zone for the report being run. This value corresponds to the Time Zone field in PeopleSoft.

Limits: Up to 9 characters; cannot contain delimiters (such as spaces)

Note: You can view the time zone settings in PeopleSoft.

JIL attribute: ps_time_zone

Chapter 18: FTP and Secure Copy Jobs

This section contains the following topics:

- [FTP Jobs](#) (see page 397)
- [Secure Copy Jobs](#) (see page 397)
- [Define a File Transfer Protocol \(FTP\) Job](#) (see page 398)
- [Define a Secure Copy Job](#) (see page 405)

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.

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 File Transfer Protocol (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 a File Transfer Protocol job

1. Create a File Transfer Protocol (FTP) job in either Quick Edit or Application Editor.

The Properties section for the FTP job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Server name

Specifies the DNS name or IP address of a remote server.

Transfer type

Specifies the type of data involved in an FTP transfer: ASCII, Auto detect, Binary, or EBCDIC (i5/OS only).

Local file name

Specifies the file's destination (if downloading) or the file's source location (if uploading).

Remote file name

Specifies the file's source location (if downloading) or the file's destination (if uploading).

3. (Optional) Specify the following property:

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 by default. The Owner property overrides the default value.

4. (Optional) Specify optional [FTP properties](#) (see page 401).
5. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
6. Commit the job.

The FTP job is defined.

More information:

[FTP Job Examples](#) (see page 399)

FTP Job Examples

The following examples describe sample FTP jobs.

Example: Download a File from a UNIX System to a Windows System

In this example, a file named `textfile` is downloaded from a UNIX system and then copied to a local Windows system. Note that each of the two locations includes a complete path statement. After the download is complete, the job completes.

To download a file from a UNIX system to a Windows system

1. Create an FTP job.
2. Enter the following properties:
 - Name—`unix_textfile`
 - Send to machine—`AGENT`
 - Server name—`HPUNIX`
 - Server port—`5222`
 - Transfer direction—`DOWNLOAD`
 - Transfer type—`ASCII`
 - Local file name—`c:\qatest\ftpdata\textfile`
 - Remote file name—`/u1/qatest/ftpdata/textfile`
3. Commit the job.

Example: Upload an ASCII-Encoded File in the Root File System from an i5/OS System to a UNIX System

In this example, a file named `textfile` in the root file system is uploaded from an i5/OS system to a UNIX system. Note that each of the two locations includes a complete path statement. After the upload is complete, the job completes.

To upload an ASCII-encoded file in the root file system from an i5/OS system to a UNIX system

1. Create an FTP job.
2. Enter the following properties:
 - Name—`i5_textfile`
 - Send to machine—I5AGENT
 - Server name—HPUNIX
 - Server port—5222
 - Transfer direction—UPLOAD
 - Transfer type—ASCII
 - Local file name—`/home/cybesp/textfile`
 - Remote file name—`/u1/qatest/ftpdata/textfile`
3. Commit the job.

Example: Download a QSYS EBCDIC-Encoded File

In this example, an EBCDIC-encoded file named datafile in the QSYS file system is downloaded from an i5/OS system to another i5/OS system. Note that the file names are specified in the path format.

To download a QSYS EBCDIC-encoded file

1. Create an FTP job.
2. Enter the following properties:
 - Name—datafile
 - Send to machine—I5AGENT
 - Server name—i5UNIX
 - Server port—5222
 - Transfer direction—DOWNLOAD
 - Transfer type—EBCDIC
 - Local file name—/QSYS.LIB/ESPLIB.LIB/DOWNLOAD.FILE/DATA.MBR
 - Remote file name—/QSYS.LIB/DATALIB.LIB/DATAFILE.FILE/DATA.MBR
3. Commit the job.

File Transfer Protocol Properties

The File Transfer Protocol category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Compression

(Optional) Specifies the data compression level.

Limits: 0-9 (0 is no data compression and 9 is the highest data compression)

Default: 0

Note: If the COMPRESS operand is not specified, the data is compressed using the level set in the ftp.data.compression parameter on the agent FTP client. This parameter is automatically set to 0 by default (no data compression).

JIL attribute: ftp_compression

Initialization command

(Optional) Defines a comma separated list of commands that are to run prior to file transfer.

Limits: Up to 1024 characters

Note: You can specify any number of initialization commands for an FTP job.

JIL attribute: ftp_command

Local file name

Specifies the file's destination (if downloading) or the file's source location (if uploading).

Limits: Up to 256 characters; case-sensitive

UNIX/Windows:

- If you are downloading a file, you must specify the full path and file name.
- If you are uploading files, you can use wildcards in the file name. The asterisk (*) is a wildcard for zero or more characters and the question mark (?) is a wildcard for a single character.
- You cannot use wildcards in the path.

i5/OS:

- To specify a file in the root file system, use UNIX path and file formats.
- To specify a *FILE object in QSYS, use the following format:

/QSYS.LIB/libraryname.LIB/objectname.FILE/membername.MBR

JIL attribute: ftp_local_name

Local user

(Optional) Specifies the user ID to be used to determine ownership of the downloaded file. The user ID must have super-EDIT authority with native security or as-owner authority if using CA EEM; otherwise, the value specified is ignored.

Limits: Up to 80 characters; case-sensitive

Notes:

- This property only applies to UNIX.
- If you do not specify a value, the authenticated user ID is used by default.

JIL attribute: ftp_local_user

Remote file name

Specifies the file's source location (if downloading) or the file's destination (if uploading).

Limits: Case-sensitive

UNIX/Windows:

- If you are uploading a file, you must specify the full path and file name.
- If you are downloading files, you can use wildcards in the file name. The asterisk (*) is a wildcard for zero or more characters and the question mark (?) is a wildcard for a single character.
- You cannot use wildcards in the path.
- On UNIX, if you want to use a Windows file as a remote file, you must use a forward slash at the beginning of the path statement and between the directories and file name, as shown in the following example:

remotefilename /C:/TEMP/textfile

i5/OS:

- To specify a file in the root file system, use UNIX path and file formats.
- To specify a *FILE object in QSYS, use the following format:

/QSYS.LIB/libraryname.LIB/objectname.FILE/membername.MBR

JIL attribute: ftp_remote_name

Server name

Specifies the DNS name or IP address of a remote server.

Limits: Up to 100 characters; case-sensitive

Example: 172.24.36.107 (IPv4) or 0:0:0:0:FFFF:192.168.00.00 (IPv6)

JIL attribute: ftp_server_name

Server port

(Optional) Specifies the port number of the remote server.

Limits: 0-65535

Default: 21

JIL attribute: ftp_server_port

Transfer direction

(Defaulted) Specifies the file transfer direction between the agent computer and the remote server using one of the following options:

- UPLOAD—Transfers files from the agent computer to the remote server.
- DOWNLOAD—Transfers files from the remote server to the agent computer.

Default: DOWNLOAD

JIL attribute: ftp_transfer_direction

Transfer type

Specifies the type of data involved in an FTP transfer: ASCII, Auto detect, Binary, or EBCDIC (i5/OS only).

Default: Binary

JIL attribute: ftp_transfer_type

Use SSL

(Optional) Indicates whether the FTP data transfer uses Secure Sockets Layer (SSL) FTP or regular FTP.

Default: unselected (uses regular FTP)

JIL attribute: ftp_use_ssl

Define a Secure Copy Job

You can define a Secure Copy job to transfer binary files using the Secure Copy Protocol (SCP) or the Secure File Transfer Protocol (SFTP).

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. Create a Secure Copy (SCP) job in either Quick Edit or Application Editor.

Note: The Secure Copy job is listed under System jobs.

The Properties section for the Secure Copy job appears.

2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Server name

Specifies the DNS name or IP address of a remote server.

Remote directory

Specifies the file's remote source directory (if downloading) or the file's remote destination directory (if uploading).

Remote file name

Specifies the file's source location (if downloading) or the file's destination (if uploading).

Local file name

Specifies the file's destination (if downloading) or the file's source location (if uploading).

Note: You can click  to open the Browse Agent File System dialog to locate the local file destination or location.

3. (Optional) Specify optional [Secure Copy properties](#) (see page 407).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The Secure Copy job is defined.

More information:

[Secure Copy Job Examples](#) (see page 406)

Secure Copy Job Examples

The following examples describe sample Secure Copy jobs.

Example: Download a File from a UNIX System to a Windows System

In this example, a file named `textfile` is downloaded from a UNIX system and then copied to a local Windows system. Note that each of the two locations includes a complete path statement. After the download is complete, the job completes.

To download a file from a UNIX system to a Windows system

1. Create a Secure Copy job.
2. Enter the following properties:
 - Name—`unix_textfile`
 - Send to machine—`AGENT`
 - Server name—`HPUNIX`
 - Transfer direction—`DOWNLOAD`
 - Remote OS type—`UNIX`
 - Remote directory—`/u1/qatest/ftpdata`
 - Remote file name—`/u1/qatest/ftpdata/textfile`
 - Local file name—`c:\qatest\ftpdata\textfile`
3. Commit the job.

Example: Upload a File Using Secure File Transfer Protocol

In this example, all files located in the ftpdata directory on a remote UNIX server are uploaded to the temporary directory on a Windows computer using SCPv2.

To upload a file using Secure File Transfer Protocol

1. Create a Secure Copy job.
2. Enter the following properties:
 - Name—wildcard
 - Send to machine—AGENT
 - Server name—UNIX
 - Transfer direction—UPLOAD
 - Remote OS type—UNIX
 - Remote directory—/u1/qatest/ftpdata
 - Remote file name—/u1/qatest/ftpdata/*
 - Local file name—c:\temp\upload*
 - Protocol—SFTP
3. Commit the job.

Secure Copy Properties

The Secure Copy category includes the following properties:

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Local file name

Specifies the file's destination (if downloading) or the file's source location (if uploading). The path should be included. The local file or files are located on the computer where the agent is installed.

Limits: Up to 256 characters; case-sensitive

Notes:

- You can use a wildcard character for the file name only if the Use SCPv2 property is selected. You cannot rename files if wildcards are used. You cannot use a wildcard character for the path.
- The semi-colon (;) cannot be used to separate filenames.

JIL attribute: scp_local_name

Local user

(Optional) Specifies a user ID on the computer where the agent is installed.

Limits: Up to 80 characters; case-sensitive

Default: The user ID that defined the Secure Copy job.

JIL attribute: scp_local_user

Protocol

(Defaulted) Specifies whether the SCP data transfer uses Secure File Transfer Protocol (SFTP) or regular Secure Copy (SCP).

Default: SFTP

JIL attribute: scp_protocol

Remote directory

Specifies the file's remote source directory (if downloading) or the file's remote destination directory (if uploading). The remote directory is not located on the computer where the agent is installed.

Limits: Up to 256 characters; case-sensitive

JIL attribute: scp_remote_dir

Remote file name

Specifies the file's source location (if downloading) or the file's destination (if uploading). The remote file or files are not located on the computer where the agent is installed.

Limits: Up to 256 characters; case-sensitive

Notes:

- You can use a wildcard character for the file name only if the Use SCPv2 property is selected. You cannot rename files if wildcards are used.
- The semi-colon (;) cannot be used to separate filenames.

JIL attribute: scp_remote_name

Remote OS type

(Defaulted) Specifies the remote OS type, which is used to determine the path separator on the remote system. The following remote OS types are supported:

- UNIX
- Windows
- OpenVMS
- Tandem
- I5/OS
- z/OS

Default: UNIX

JIL attribute: scp_target_os

Server name

Specifies the DNS name or IP address of a remote server.

Limits: Up to 256 characters; case-sensitive

JIL attribute: scp_server_name

Server port

(Optional) Specifies the port number of the remote server.

Limits: 0-65535

Default: 22

JIL attribute: scp_server_port

Transfer direction

(Defaulted) Select the file transfer direction between the agent computer and the remote server, as follows:

- UPLOAD—Transfers files from the agent computer to the remote server.
- DOWNLOAD—Transfers files from the remote server to the agent computer.

Default: DOWNLOAD

JIL attribute: scp_transfer_direction

Chapter 19: i5/OS Jobs

This section contains the following topics:

- [i5/OS Jobs](#) (see page 411)
- [Running UNIX Workload on a System i5 Computer](#) (see page 412)
- [Define an i5/OS Job](#) (see page 413)
- [Pass Positional Parameters](#) (see page 423)
- [Use a User's Library List](#) (see page 423)
- [Pass Keyword Parameters to SBMJOB](#) (see page 424)
- [Returning a Job's Exit Status to CA Workload Automation AE](#) (see page 425)
- [Specify Data for a Local Data Area](#) (see page 427)

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 *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.

Note: To run these jobs, your system requires CA WA Agent for i5/OS.

To define an i5/OS job

1. Create an i5/OS (I5) job in either Quick Edit or Application Editor.
The Properties section for the i5/OS job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Send to machine

Specifies the name of the computer where the agent runs.

Action (CMD/Program) name

Specifies the program, the source file for the program, or the command that you want to run.

Notes:

- The value must correspond to the action value. If you do not specify the action value, the job interprets the corresponding Action (CMD/Program) name value as a command by default.
- You can click  to open the Browse Agent File System dialog to locate the program, source file for the program, or the command on the agent computer.

3. (Optional) Specify optional [i5/OS properties](#) (see page 418).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The i5/OS job is defined.

More information:

[Running UNIX Workload on a System i5 Computer](#) (see page 412)
[i5/OS Naming Conventions](#) (see page 412)
[Pass Positional Parameters](#) (see page 423)
[Use a User's Library List](#) (see page 423)
[Pass Keyword Parameters to SBMJOB](#) (see page 424)
[Returning a Job's Exit Status to CA Workload Automation AE](#) (see page 425)
[Specify Data for a Local Data Area](#) (see page 427)
[i5/OS Properties](#) (see page 418)
[i5/OS Job Examples](#) (see page 414)

i5/OS Job Examples

The following examples describe sample i5/OS jobs.

Example: Run an i5/OS Command

This example runs the command named CALC on the i5agent computer.

To run an i5/OS command

1. Create an i5/OS job.
2. Enter the following properties:
 - Name—i5job_runcmd
 - Send to machine—i5agent
 - Action (CMD/Program) name—CALC
3. Commit the job.

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 single quotation marks because it contains a space.

To pass multiple parameters to an i5/OS job

1. Create an i5/OS job.
2. Enter the following properties:
 - Name—i5job_lib
 - Send to machine—i5agent
 - Action—RUN_PROGRAM
 - Action (CMD/Program) name—PAYJOB
 - Positional parameters—"ABC 1 P 'VALUE C' X r"
3. Commit the job.

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.

To specify the printer and output queue for an i5/OS job

1. Create an i5/OS job.
2. Enter the following properties:
 - Name—i5job_lib
 - Send to machine—i5agent
 - Action—RUN_PROGRAM
 - Action (CMD/Program) name—PAYJOB
 - Keyword parameters—PRTDEV=*JOBD,OUTQ=*JOBD
3. Commit the job.

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.

To send an OPM COBOL program's return code as the job's exit code

1. Create an i5/OS job.
2. Enter the following properties:
 - Name—i5job_returnOPM
 - Send to machine—i5agent
 - Action—RUN_PROGRAM
 - Action (CMD/Program) name—PAYROLL
 - i5/OS exit program—*PROGRAM
3. Commit the job.

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 0 (zero) or 10 indicate job success.

To send an ILE C program's return code as the job's exit code

1. Create an i5/OS job.
2. Enter the following properties:
 - Name—i5job_returnOPM
 - Send to machine—i5agent
 - Action—RUN_PROGRAM
 - Action (CMD/Program) name—SALARY
 - i5/OS exit program—*USER
3. Enter the following Termination property:
 - Success exit codes—0,10
4. Commit the job.

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 0 or 100.

To send exit code 100 as success

1. Create an i5/OS job.
2. Enter the following properties:
 - Name—i5job_succ
 - Send to machine—i5agent
 - Action—RUN_PROGRAM
 - Action (CMD/Program) name—PAYPROG
 - i5/OS exit program—*PROGRAM
3. Enter the following Termination property:
 - Success exit codes—0,100
4. Commit the job.

Example: Send Exit Code 40 as Failure

This example runs the RECPROG program. The program is considered to have failed if it returns an exit code of 40. All other exit codes in the range from 0 to 255 indicate job success.

To send exit code 40 as failure

1. Create an i5/OS job.
2. Enter the following properties:
 - Name—i5job_fail
 - Send to machine—i5agent
 - Action—RUN_PROGRAM
 - Action (CMD/Program) name—RECPROG
 - i5/OS exit program—*PROGRAM
3. Enter the following Termination properties:
 - Success exit codes—0-255
 - Failure exit codes—40
4. Commit the job.

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 hexadecimal data. When the job completes, the local data area is destroyed automatically by the operating system.

To specify data for the local data area in hexadecimal format

1. Create an i5/OS job.
2. Enter the following properties:
 - Name—i5job_lda
 - Send to machine—i5agent
 - Action—COMMAND
 - Action (CMD/Program) name—IVP
 - Local data area—X'C1C2C3C4'
3. Commit the job.

i5/OS Properties

The i5/OS category includes the following properties:

Action

(Defaulted) Specifies whether to run a program or issue a command on an i5/OS system, as follows:

- RUN_PROGRAM—Runs a program on an i5/OS system.
- RUN_PROG_IN_FILE—Runs a program from a database file member on an i5/OS system. This is similar to running a shell script on a UNIX system or a batch file on a Windows system.
- COMMAND—Issues a command on an i5/OS system when the i5/OS job runs.

Default: COMMAND

Note: When the RUN_PROG_IN_FILE option is selected, the pass positional parameters property is disabled.

JIL attribute: i5_action

Action (CMD/Program) name

Specifies the program, the source file for the program, or the command that you want to run.

Limits: Up to 120 characters for command action, up to 60 characters for other i5 actions; case-sensitive

Note: You can click  to open the Browse Agent File System dialog to locate the program, source file for the program, or the command on the agent computer.

JIL attribute: i5_name

Agent job class

(Optional) Specifies the job class that this job runs under. The agent maintains a list of job classes and the number of initiators assigned to each job class. A job class with more initiators can process a greater number of jobs more quickly. For higher-priority jobs, assign a job class that contains more initiators.

Limits: Up to 32 characters; cannot contain delimiters (such as spaces)

JIL attribute: job_class

Current library

(Optional) Specifies the current library for the job.

Limits: Up to 46 characters; cannot contain delimiters (such as spaces)

JIL attribute: i5_curr_lib

i5/OS exit program

(Defaulted) Specifies the type of exit code returned by an i5/OS job, as follows:

- *SEVERITY—Sends the job's ending severity code.
- *USER—Sends the return code of an ILE program or module. For example, if your job runs an ILE C or ILE RPG program that contains an exit or return statement, that return code is sent as the exit code.
- *PROGRAM—Sends the return code of an OPM program. For example, if your job runs an OPM RPG or OPM COBOL program that contains an exit or return statement, that return code is sent as the exit code.

Default: *SEVERITY

JIL attribute: i5_cc_exit

i5/OS job description

(Optional) Specifies the job description for the submitted program using one of the following formats:

description
library/*description*
/QSYS.LIB/*library*.LIB/*description*.JOBD

description

Specifies the name of the job description for the program submitted. The value must be a valid i5/OS job description name.

library

Specifies the name of the library that contains the job description. The value must be a valid i5/OS library name.

Limits: Up to 256 characters; cannot contain delimiters (such as spaces)

Note: You can click  to open the Browse Agent File System dialog to locate the job description on the agent computer.

JIL attribute: i5_job_desc

i5/OS job name

(Optional) Specifies the name of the i5/OS job.

Limits: Up to 10 characters; the first character must be alphabetical

JIL attribute: i5_job_name

i5/OS job queue

(Optional) Specifies the i5/OS job queue for the submitted program using one of the following formats:

jobqueue
library/*jobqueue*
/QSYS.LIB/*library*.LIB/*jobqueue*.JOBQ

jobqueue

Specifies the name of the job queue for the submitted program. The value must be a valid i5/OS job queue name.

library

Specifies the name of the library that contains the job description. The value must be a valid i5/OS library name.

Limits: Up to 46 characters; case-sensitive

Note: You can click  to open the Browse Agent File System dialog to locate the i5/OS job queue on the agent computer.

JIL attribute: i5_job_queue

i5/OS process priority

(Defaulted) Specifies the the order in which processes are scheduled on the System i5 processor, as follows:

- High—Indicates processes that must be executed immediately. These processes can use nearly all available CPU time.
- Above normal—Indicates processes that have priority above the Normal level, but below the High level.
- Normal—Indicates processes without special scheduling needs.
- Below normal—Indicates processes that have priority above the Idle level, but below the Normal level.
- Idle—Indicates processes that will run only when the system is idle.

Depending on the priority level, process priority can speed up or slow down a process.

Default: Normal

JIL attribute: i5_process_priority

Keyword parameters

(Optional) Specifies any valid SBMJOB command keyword and value combination. The value corresponds to the SBMJOB command keyword (the value that would appear in brackets after the keyword in the SBMJOB command). Code parameters using

Limits: Up to 4096 characters; enclose values that contain spaces in single quotation marks

Notes:

- You can specify multiple keyword=value combinations. Separate each combination with a comma.
- Click  to open the Edit Text dialog.

JIL attribute: i5_others

Library

(Optional) Specifies the name of the library that contains the program, the source file for the program, or the command that you want to run. The value must be a valid i5/OS library name.

Limits: Up to 46 characters; cannot contain delimiters (such as spaces)

Note: Instead of using the Library field, you can specify the library name in the i5/OS (CMD/Program) name field, or you can specify the library name using the Current library or Library list fields.

JIL attribute: i5_lib

Library list

(Optional) Specifies the name of a library that you want to add to the library list.

Limits: Up to 274 characters; cannot contain delimiters (such as spaces)

JIL attribute: i5_library_list

Local data area

(Optional) Specifies data for the local data area in an i5/OS job using the following format:

"*data*" | X'hh...'

"*data*"

Specifies the data in character format.

Example: "My data"

X'hh...'

Specifies the data in hexadecimal format, with each pair of hexadecimal digits representing one byte of data.

Example: X'C1C2C340C1C2C3'

Limits: Up to 2051 characters; case-sensitive

Notes:

- The local data area is a temporary 1024-byte storage area that exists for the duration of the 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.

■ Click  to open the Edit Text dialog.

JIL attribute: i5_lda

Positional parameters

(Optional) Defines the parameter values that you want to pass to the program at the time the program is invoked.

Limits: Up to 512 characters; enclose parameters in double quotes; enclose parameters containing a space in single quotes

Note: Click  to open the Edit Text dialog.

JIL attribute: i5_params

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 413).
2. Add the following property to the job definition:

Positional parameters

Defines the parameter values that you want to pass to the program at the time the program is invoked.

3. Commit and run the job.

The specified positional parameters are passed to the i5/OS program.

Use a User's Library List

The agent uses the library list in the i5/OS 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 413).
2. Do *one* of the following:
 - Specify the following i5/OS property:

Current library

Specifies that the job uses the user's current library when it runs.

- Specify the following i5/OS property:

Library

Specifies the name of the library that contains the program, the source file for the program, or the command that you want to run.

- Specify the following i5/OS properties:

Library list

Specifies the name of a library that you want to add to the library list.

i5/OS job description

Specifies the job description for the submitted program.

3. Commit and run the job.

The job uses the user's library list.

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 i5/OS properties map to parameters for the SBMJOB command:

| i5/OS Properties | SBMJOB Parameters |
|-----------------------|-------------------|
| i5/OS job description | JOBD |
| Library list | INLLIBL |
| i5/OS job queue | JOBQ |
| Current library | CURLIB |
| i5/OS 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 413).
2. Add the following attribute to the job definition:

Keyword parameters

Specifies SBMJOB command keyword and value combinations.

3. Commit and run the job.

The specified keywords and values are passed to the SBMJOB command.

Note: The special values for these SBMJOB parameters, such as *USRPRF and *JOBD, also apply to the properties. 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.

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.

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/OS exit program i5/OS property
- Send a user-defined exit code using the Success exit codes or Failure exit codes Termination properties

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 an ILE program or module or 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 413).
2. Select *one* of the following i5/OS exit program options:

*SEVERITY

Sends the job's ending severity code.

*USER

Specifies that the return code of an ILE program or module is sent as the exit code.

*PROGRAM

Specifies that the return code of an OPM program is sent as the exit code.

3. Commit and run the job.

The program's return code is sent instead of the job's ending severity code.

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 exit codes property in the Termination section of the job definition.

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 413).
2. Add *one* of the following Termination properties to the job definition:

Success exit codes

Defines which exit codes indicate job success.

Default: 0 (zero)

Failure exit codes

Defines which exit codes indicate job failure.

Default: Any non-zero exit code

3. Commit and run the job.

The specified user-defined exit code is sent to CA Workload Automation AE.

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 413).
2. Add the following property to the job definition:

Local data area

Specifies data for the local data area in an i5/OS job.

3. Commit and run the job.

The data is specified for a local data area.

Chapter 20: z/OS Jobs

This section contains the following topics:

- [z/OS Jobs](#) (see page 429)
- [z/OS Data Set Trigger Jobs](#) (see page 430)
- [Define a z/OS Data Set Trigger Job](#) (see page 431)
- [Define a z/OS Manual Job](#) (see page 439)
- [Define a z/OS Regular Job](#) (see page 441)

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.

z/OS Data Set Trigger Jobs

You can define z/OS Data Set Trigger jobs to create 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. Create a z/OS Data Set Trigger (ZOSDST) job in either Quick Edit or Application Editor.
The Properties section for the z/OS Data Set Trigger job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

Data set name

Specifies the Job Control Language (JCL) library name.

3. (Optional) Specify optional [z/OS Data Set Trigger properties](#) (see page 436).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The z/OS Data Set Trigger job is defined.

More information:

[z/OS Data Set Trigger Job Examples](#) (see page 432)

z/OS Data Set Trigger Job Examples

The following examples describe sample z/OS Data Set Trigger jobs.

Example: Monitor for Data Set Creation and Update

Suppose that you want a z/OS Data Set Trigger job named PROD.NIGHTLY to release its successors when the data set PROD.CICS.FILE1602 is closed (created or updated). The agent ZOS1 monitors the data set under user CYBDL01.

To monitor for data set creation and update

1. Create a z/OS Data Set Trigger job.
2. Enter the following properties:
 - Name—PROD.NIGHTLY
 - Send to machine—ZOS1
 - Owner—CYBDL01
 - Data set name—PROD.CICS.FILE1602
3. Save the job.

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.

To restrict the trigger to specific data sets created by a particular job

1. Create a z/OS Data Set Trigger job.
2. Enter the following properties:
 - Name—PROD.PAY_DATA
 - Send to machine—ZOS1
 - Owner—CYBDL01
 - Data set name—USER1.PAYROLL.G-
 - Trigger by—Job name, ABC
3. Save the job.

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.

To restrict the trigger to specific data sets created by a particular user

1. Create a z/OS Data Set Trigger job.
2. Enter the following properties:
 - Name—PROD.PAY_DATA
 - Send to machine—ZOS1
 - Owner—CYBDL01
 - Data set name—USER1.PAYROLL.G-
 - Trigger by—user ID, CYB1
3. Save the job.

Example: Run a Compress Job After 100 Closures of a Data Set

Suppose that you want the z/OS Data Set Trigger job PROD.PAY_DATA to release a compress job named COPYJCL.COMPRESS after every 100 closures of the data set CYBER.COPY.JCL. The agent ZOS1 monitors the data set under user CYBDL01.

To run a compress job after 100 closures of a data set

1. Create a z/OS Data Set Trigger job.
2. Enter the following properties:
 - Name—PROD.PAY_DATA
 - Send to machine—ZOS1
 - Owner—CYBDL01
 - Data set name—CYBER.COPY.JCL
 - Trigger on—100
3. Save the job.

Note: Define the compress job, COPYJCL.COMPRESS, as a successor to the z/OS Data Set Trigger job.

Example: Monitor for a Data Set Received from a Remote FTP Partner

Suppose that you want the z/OS Data Set Trigger job PROD.PAY_DATA to release its successors when a file is successfully received from a remote FTP partner, creating generation data set USER1.PAYROLL (USER1.PAYROLL.G-). The agent ZOS1 monitors the FTP transfer under user CYBDL01.

To monitor for a data set received from a remote FTP partner

1. Create a z/OS Data Set Trigger job.
2. Enter the following properties:
 - Name—PROD.PAY_DATA
 - Send to machine—ZOS1
 - Owner—CYBDL01
 - Data set name—USER1.PAYROLL.G-
 - FTP direction—RECEIVE
3. Save the job.

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.

To monitor for a data set sent to a remote FTP partner

1. Create a z/OS Data Set Trigger job.
2. Enter the following properties:
 - Name—CYBER.XFER
 - Send to machine—ZOS1
 - Owner—CYBDL01
 - Data set name—CYBER.XFER.001
 - FTP direction—SEND
3. Save the job.

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.

To restrict triggering to a specific host

1. Create a z/OS Data Set Trigger job.
2. Enter the following properties:
 - Name—CYBER.XFER
 - Send to machine—ZOS1
 - Owner—CYBDL01
 - Data set name—CYBER.XFER.001
 - FTP direction—RECEIVE
 - FTP host—172.16.0.0
3. Save the job.

Example: Restrict Triggering to a Specific User 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 user ID prefix of the local FTP partner is CYB (CYB-). The agent ZOS1 monitors the FTP transfer under user CYBDL01.

To restrict triggering to a specific user ID

1. Create a z/OS Data Set Trigger job.
2. Enter the following properties:
 - Name—CYBER.XFER
 - Send to machine—ZOS1
 - Owner—CYBDL01
 - Data set name—CYBER.XFER.001
 - Trigger by—user ID, CYB-
 - FTP direction—RECEIVE
3. Save the job.

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.

To restrict triggering to a specific logon ID

1. Create a z/OS Data Set Trigger job.
2. Enter the following properties:
 - Name—CYBER.XFER
 - Send to machine—ZOS1
 - Owner—CYBDL01
 - Data set name—CYBER.XFER.001
 - FTP direction—RECEIVE
 - FTP userid—CYBER005
3. Save the job.

z/OS Data Set Trigger Properties

The z/OS Data Set Trigger category includes the following properties:

Data set name

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. Data set names typically have up to four positions; positions must each be eight characters or less and separated by periods.

Limits: Up to 44 characters

Example: prod.jcllib

JIL attribute: zos_dataset

FTP direction

(Optional) Specifies the direction of FTP transfer between a remote computer and a local mainframe computer, as follows:

- RECEIVE—Transfers from a remote computer to a local mainframe computer.
- SEND—Transfers to a remote computer from a local mainframe computer.

Note: This field is valid only when the Trigger on explicit data set value is False.

JIL attribute: zos_ftp_direction

FTP host

(Optional) Specifies the host name used to restrict the activation of the data set trigger from transferring to or from the specified remote host.

Limits: Up to 128 characters

Note: This field is valid only when the Trigger on explicit data set value is False.

JIL attribute: zos_ftp_host

FTP userid

(Optional) Specifies the user ID for the FTP connection.

Limits: Up to 80 characters

Note: This field is valid only when the Trigger on explicit data set value is False.

JIL attribute: zos_ftp_userid

Trigger by

(Optional) Specifies whether the job monitors data set activity by a job or a user ID. Specify the name of the job or user responsible for the data set activity that triggers the job in the Type value field. When a data set is updated or renamed by the specified job or user ID, the job triggers.

Limits: Up to 128 characters

JIL attribute: zos_trigger_type, zos_trigger_by

Trigger on

(Optional) Specifies the number of actions that must occur before the job triggers.

Limits: 0-100

JIL attribute: zos_trigger_on

Trigger on data set renamed

(Optional) Indicates whether the job monitors when a data set is renamed. If selected, when the data set is renamed, the job triggers.

JIL attribute: zos_dsn_renamed

Trigger on data set updated

(Optional) Indicates whether the job monitors when a data set is updated. If selected, when the data set is updated, the job triggers.

JIL attribute: zos_dsn_updated

Trigger on explicit data set

(Optional) Indicates whether the job monitors for an explicit data set notification (used when the data set activity does not generate an SMF record). When selected, the job triggers when it receives notification that the specified data set is being updated.

Default: unselected (the job does not monitor for an explicit data set notification)

JIL attribute: zos_explicit_dsn

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. Create a z/OS Manual (ZOS) job in either Quick Edit or Application Editor.
The Properties section for the z/OS Manual job appears.
2. Specify the following required properties:

Name

Defines the name of the job submitted outside the scheduling manager.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

3. (Optional) Specify optional [z/OS Manual properties](#) (see page 440).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The z/OS Manual job is defined.

More information:

[z/OS Manual Job Examples](#) (see page 440)

z/OS Manual Job Examples

The following example describes a sample z/OS Manual job.

Example: Post a z/OS Manual Job as Complete Based on the User ID

Suppose that you want to post a z/OS Manual job to complete when the manually-submitted job ABC runs under user CYBER. The ZOS1 agent monitors job ABC.

To post a z/OS Manual job as complete based on the user ID

1. Create a z/OS Manual job.
2. Enter the following properties:
 - Name—ABC
 - Send to machine—ZOS1
 - Authorization string—CYBER
3. Commit the job.

z/OS Manual Properties

The z/OS Manual category includes the following properties:

Authorization string

(Optional) Defines the authorization string the scheduling manager uses to post and track the correct job.

Limits: Up to 80 characters

Note: Contact your agent administrator to confirm whether the agent checks the authorization string. Depending on the scenario, the action to take differs as follows:

- If authorization checking (AUTHSTR) is not set up for this agent, do not specify the authorization string.
- If authorization checking (AUTHSTR) is set to RACUSER, specify the user ID that owns the job.
- If authorization checking (AUTHSTR) is set to ACCOUNT1, ACCOUNT2, ACCOUNT3, or ACCOUNT4, specify the value found in the corresponding position of the job's Job Control Language (JCL). For example, if AUTHSTR is set to check the ACCOUNT2 field, specify the value found in the second accounting position in the job's JCL.

JCL attribute: auth_string

Backward search (hours)

(Optional) Defines the number of hours that the agent does a backward search for a manually-submitted job.

Limits: 0-999

JIL attribute: search_bw

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. Create a z/OS Regular (ZOS) job in either Quick Edit or Application Editor.
The Properties section for the z/OS Regular job appears.
2. Specify the following required properties:

Name

Defines the name of the job that you want to schedule.

Limits: Up to 64 characters; valid characters are a-z, A-Z, 0-9, period (.), underscore (_), hyphen (-), and pound (#); do not include embedded spaces or tabs

Send to machine

Specifies the name of the computer where the agent runs.

JCL library

Specifies the library name containing the Job Control Language (JCL) for this job.

Note: If you specify a partitioned data set, you must also specify the jcl_member attribute in your job definition.

3. (Optional) Specify optional [z/OS Regular properties](#) (see page 442).
4. (Optional) Specify [common properties that apply to all job types](#) (see page 39).
5. Commit the job.

The z/OS Regular job is defined.

More information:

[z/OS Regular Job Examples](#) (see page 442)

z/OS Regular Job Examples

The following example describes a sample z/OS Regular job.

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.

To store a working copy of the JCL that you submitted

1. Create a z/OS Regular job.
2. Enter the following properties:
 - Name—CYBDL01A
 - Send to machine—ZOS1
 - JCL library—CYBDL01.JCLLIB
 - JCL member—CYBDL01A
 - Copy JCL library—CYBDL01.COPY.JCLLIB
3. Commit the job.

z/OS Regular Job Properties

The z/OS Regular category includes the following properties:

Condition code: Return code

Specifies the return code using one of the following values:

- A condition code for one number or a range of numbers between 1 and 4095 (ranges are separated by colons, such as 1:4095).
- A system abend code (Sccc), such as S0C1 or SB37.
- A user abend code (Unnnn), such as U0001 or U0462. The nnnn must be exactly four decimal digits and cannot exceed 4095.

Limits: Up to 9 characters

JIL attribute: condition_code: rc

Condition code: Result

Specifies whether to consider the job as failed based on the condition code, as follows:

- SUCCESS—Considers the job not failed if the condition code is matched.
- FAILURE—Considers the job failed if the condition code is matched.

Default: SUCCESS

JIL attribute: result

Condition code: Action

Specifies whether the job continues with the next step regardless of whether the job is considered as failed.

- CONTINUE—Continues with the next step if the condition code is matched.
- STOP—Stops the job if the condition code is matched.

Default: CONTINUE

JIL attribute: action

Condition code: Procstep

Specifies a valid z/OS procedure name.

Limits: Up to 8 characters; first character must not be numeric

JIL attribute: procstep

Condition code: Program

Specifies a valid z/OS program name.

Limits: Up to 8 characters; first character must not be numeric

JIL attribute: program

Condition code: Stepname

Specifies a valid z/OS step name.

Limits: Up to 8 characters; first character must not be numeric

JIL attribute: stepname

Copy JCL library

(Optional) Specifies the data set name that receives the working copy of the JCL you submitted. Data set names typically have up to four positions; positions must each be eight characters or less and separated by periods.

Limits: Up to 44 characters

Example: prod.copyjcl.weekly

JIL attribute: copy_jcl

JCL library

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. Data set names typically have up to four positions. Each position can be up to eight characters. Separate the positions using periods. The JCL library can be any one of the following:

- A valid z/OS data set name. You can specify a partitioned data set (PDS) or a sequential data set.
- A valid z/OS data set name, prefixed with pan- to indicate Panvalet input
- A valid z/OS data set name, prefixed with lib- to indicate Librarian input

Limits: Up to 44 characters

Note: If you specify a partitioned data set, you must also specify the jcl_member attribute in your job definition.

JIL attribute: jcl_library

JCL member

(Optional) Specifies the JCL member that contains the JCL for your job.

Limits: Up to 8 characters

JIL attribute: jcl_member

Index

A

agent plug-in, description • 12

agent, description • 12

Application Editor

defining jobs • 25

purpose • 11

B

Box job

adding a job • 89

defining • 86

description • 85

grouping • 88

properties • 87

Box properties, field descriptions • 87

C

Command jobs

checking available file space • 102

defining • 92

description • 91

passing data to a command or script • 103

properties • 98

redirecting output • 106

running frequently used commands • 109

UNIX examples • 93

Windows examples • 96

CPU Monitoring jobs • 241

defining • 244

description • 241

examples • 248

properties • 248

custom calendar, description • 15

cycle

creating • 17

description • 15

example • 17

D

Database Monitor jobs

comparison to Database • 204

defining • 205

description • 203

examples • 206

properties • 207

Database Stored Procedure jobs

data types supported by database • 217

defining • 209

description • 203

examples • 210

properties • 214

Database Trigger jobs

defining • 219

description • 203

examples for IBM DB2 • 227

examples for Microsoft SQL Server • 224

examples for Oracle • 220

properties • 228

date and time dependencies

custom calendars • 54

examples • 58

must start and must complete times • 55

time zone • 54

Disk Monitoring jobs

defining • 252

description • 241

examples • 253

properties • 257

documentation, available for CA Workload Automation AE • 13

E

Entity Bean jobs

defining • 122

description • 119

examples • 123

properties • 125

environment variables

description • 63

examples • 63

for UNIX systems • 105

extended calendar

creating • 19

description • 15

properties • 21

F

field descriptions • 67, 83, 98, 110

File Trigger jobs

-
- defining • 261
 - description • 242
 - examples • 262
 - properties • 266
 - File Watcher jobs
 - defining • 116
 - description • 115
 - examples • 117
 - properties • 117
 - FTP jobs
 - defining • 398
 - description • 397
 - examples • 399
 - properties • 401
- H**
- HTTP jobs
 - defining • 181
 - description • 177
 - examples • 182
 - properties • 183
- I**
- i5/OS jobs
 - defining • 413
 - description • 411
 - examples • 414
 - i5/OS naming conventions • 412
 - passing keyword parameters to SBMJOB • 424
 - passing variable data to an i5/OS program • 423
 - properties • 418
 - returning a job's exit status • 425
 - running UNIX workload in a PASE environment • 412
 - sending a program's return code • 425
 - sending a user-defined exit code • 426
 - specifying data for a local data area • 427
 - specifying the user's library list • 423
 - IP Monitoring jobs
 - defining • 271
 - description • 241
 - examples • 272
 - properties • 272
- J**
- JMS Publish jobs
 - defining • 141
 - JMS Subscribe jobs
 - description • 137
 - examples • 142
 - properties • 143
 - JMX-MBean Attribute Get jobs
 - defining • 146
 - description • 137
 - examples • 147
 - properties • 147
 - JMX-MBean Attribute Set jobs
 - defining • 154
 - description • 151
 - examples • 155
 - properties • 155
 - JMX-MBean Create Instance jobs
 - defining • 161
 - description • 151
 - examples • 162
 - properties • 162
 - JMX-MBean Operation jobs
 - defining • 164
 - description • 151
 - examples • 165
 - properties • 165
 - JMX-MBean Remove Instance jobs
 - defining • 168
 - description • 151
 - examples • 169
 - properties • 169
 - JMX-MBean Subscribe jobs
 - defining • 171
 - description • 151
 - examples • 172
 - properties • 173
 - job
 - date and time dependencies • 53
 - description • 23
 - environment variables • 63
 - notifications • 75
 - resource dependencies • 50
 - start conditions • 40
 - termination conditions • 70
 - user permissions • 82
 - job flow
 - creating • 25

-
- description • 25
 - job owner, specifying • 39
 - job profile
 - description • 63
 - examples • 66
 - job termination conditions
 - automatic deletion on job completion • 72
 - exit codes • 70
 - failed box job or containing box • 73
 - maximum runtime • 72
- N**
- Notification properties, field descriptions • 79
 - notifications
 - alarms • 76
 - email • 77
 - service desk request • 78
- O**
- Oracle E-Business Suite Copy Single Request jobs
 - defining • 298
 - description • 297
 - examples • 299
 - properties • 300
 - Oracle E-Business Suite Request Set jobs
 - defining • 301
 - description • 297
 - examples • 306
 - properties • 306
 - Oracle E-Business Suite Single Request jobs
 - defining • 310
 - description • 297
 - examples • 311
 - properties • 312
- P**
- payload producing job, description • 152
 - permissions, specifying within a job definition • 82
 - POJO jobs
 - defining • 188
 - description • 178
 - examples • 189
 - properties • 189
 - Primary properties, field descriptions • 47
 - Process Monitoring jobs
 - defining • 274
 - description • 241
- examples • 275
 - properties • 277
- Q**
- Quick Edit
 - creating a global variable • 36
 - creating a job • 35
 - purpose • 11
 - saving changes • 37
- R**
- resource dependencies
 - examples • 52
 - specifying • 50
 - RMI jobs
 - defining • 191
 - description • 179
 - examples • 192
 - properties • 192
- S**
- SAP Batch Input Session jobs
 - copying a job from the SAP system • 323
 - creating a filter • 320
 - defining • 324
 - defining print or archive parameters • 370
 - defining recipients for job output • 367
 - description • 318
 - examples • 325
 - listing • 319, 322
 - properties • 325
 - sending the SAP spool file • 380
 - SAP Business Warehouse InfoPackage jobs
 - copying a job from the SAP system • 323
 - creating a filter • 320
 - defining • 329
 - description • 318
 - examples • 330
 - listing • 319, 322
 - properties • 330
 - SAP Business Warehouse Process Chain jobs
 - copying a job from the SAP system • 323
 - creating a filter • 320
 - defining • 333
 - description • 318
 - examples • 334
 - listing • 319, 322
 - properties • 334
 - SAP Data Archiving jobs

-
- archiving parameters • 339
 - defining • 336
 - description • 318
 - examples • 337
 - print parameters • 342
 - properties • 338
 - SAP Event Monitor jobs
 - defining • 346
 - description • 318
 - examples • 347
 - properties • 348
 - SAP Job Copy jobs
 - defining • 350
 - description • 318
 - examples • 351
 - properties • 351
 - SAP Process Monitor jobs
 - defining • 354
 - description • 318
 - examples • 355
 - properties • 357
 - SAP R/3 jobs
 - copying a job from the SAP system • 323
 - creating a filter • 320
 - defining • 360
 - defining print or archive parameters • 370
 - defining recipients for job output • 367
 - description • 318
 - examples • 361
 - listing • 319, 322
 - properties • 363
 - sending the SAP spool file • 380
 - using success and failure messages • 356
 - Schedule properties, field descriptions • 60
 - Secure Copy jobs
 - defining • 405
 - description • 397
 - examples • 406
 - properties • 407
 - Session Bean jobs
 - defining • 129
 - description • 120
 - examples • 130
 - properties • 131
 - SQL jobs
 - defining • 231
 - description • 203
 - examples for IBM DB2 • 236
 - examples for MicroSoft SQL Server • 234
 - examples for Oracle • 232
 - properties • 238
 - standard calendar
 - creating • 16
 - description • 15
 - starting conditions
 - cross-instance job dependencies • 42
 - examples • 45
 - exit code dependencies • 42
 - global variable dependencies • 44
 - job status dependencies • 41
 - look-back dependencies • 43
- T**
- Termination properties, field descriptions • 73
 - Text File Reading and Monitoring jobs
 - defining • 279
 - description • 241
 - examples • 280
 - properties • 282
- U**
- User Defined jobs
 - defining • 110
 - description • 109
 - properties • 110
 - UWCC, example, running workload using jobs • 12
 - UWCC--UG--PeopleSoft jobs
 - defining • 382
 - description • 381
 - examples • 384
 - properties • 388
- W**
- Web Service jobs
 - defining • 194, 195
 - description • 180
 - examples • 196
 - properties • 198
 - Windows Event Log Monitoring jobs
 - defining • 286
 - description • 241
 - examples • 287
 - properties • 288
 - Windows Service Monitoring jobs
 - defining • 292
 - description • 241
 - examples • 293

properties • 293

Z

z/OS Data Set Trigger jobs

defining • 431

description • 429

examples • 432

properties • 436

z/OS Manual jobs

defining • 439

description • 429

examples • 440

properties • 440

z/OS Regular jobs

defining • 441

description • 429

examples • 442

properties • 442