CA Workload Automation DE

Examples Cookbook

r11.3

CA Technologies Product References

This document references the following CA Technologies products:

- CA Workload Automation DE
- CA Workload Automation Desktop Client (CA WA Desktop Client)
- CA Workload Automation DE Web Client
- CA Workload Automation High Availability DE (CA WA High Availability)
- CA Workload Automation Web Services (CA WA Web Services)
- CA Workload Automation Agent for UNIX (CA WA Agent for UNIX)
- CA Workload Automation Agent for Linux (CA WA Agent for Linux)
- CA Workload Automation Agent for Windows (CA WA Agent for Windows)
- CA Workload Automation Agent for i5/OS (CA WA Agent for i5/OS)
- CA Workload Automation Agent for z/OS (CA WA Agent for z/OS)
- CA Workload Automation Agent for Application Services (CA WA Agent for Application Services)
- CA Workload Automation Agent for Web Services (CA WA Agent for Web Services)
- CA Workload Automation Agent for Micro Focus (CA WA Agent for Micro Focus)
- CA Workload Automation Agent for Databases (CA WA Agent for Databases)
- CA Workload Automation Agent for SAP (CA WA Agent for SAP)
- CA Workload Automation Agent for PeopleSoft (CA WA Agent for PeopleSoft)
- CA Workload Automation Agent for Oracle E-Business Suite (CA WA Agent for Oracle E-Business Suite)
- CA Workload Automation Restart Option EE (CA WA Restart Option)
- CA Spectrum® Service Assurance (CA Spectrum SA)

Contact CA Technologies

Contact CA Support

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

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

Providing Feedback About Product Documentation

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

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

Contents

Chapter 1: Using this Guide

The *Examples Cookbook* contains real-life examples of using CA Workload Automation DE, which you can adapt to meet your own business requirements. Many of the examples were developed for customers and are being used in their installations.

Copy and Paste Code from the PDF

The code in the examples is meant to be copied and used as needed. You can use the code in the examples by copying and pasting it from the PDF. If you need to download the Adobe Acrobat PDF reader, you can download it for free from the Adobe website (http://www.adobe.com).

Follow these steps:

1. Select the Text Select Tool from the Basic Tools toolbar.

   The Text Select Tool is activated.

2. Select the text you want to copy and press Ctrl+C.

   The selected text is copied to the clipboard.

3. Press Ctrl+V in the application you want to paste the text into.

   The selected text is pasted into your application.

Intended Audience

The examples in this document are intended for users who have a basic working knowledge of CA Workload Automation DE. A wide range of examples is provided and will be of use to beginner through to advanced users.

Many of the examples in this document use JavaScript scripts. For information about using JavaScript scripts with CA Workload Automation DE, see the *Programming Guide*.

Case Sensitivity

Schedule criteria is not case sensitive. The following are equivalent:

- LAST SATURDAY OF MONTH
- Last Saturday of Month
- last Saturday of month
- last saturday of month

The examples of schedule criteria in this book use lowercase.

Note: Many examples in this document use JavaScript script. JavaScript script is case sensitive. Be sure to use the correct case when creating JavaScript statements.

Chapter 2: Examples

Scheduling a job on, or after, a criteria

Objective

Run a job on the 8th day of the month, if this is a Sunday. Otherwise, run the job on the first Sunday after the 8th.

Solution

Use the following run criteria for the job:

```
sunday 8th monthly
```

Explanation

This solution runs the job on the first Sunday on or after the 8th day of each month.

Scheduling a job within a month

Objective

Run a job on the 5th Friday of the month but only in months that have 5 Fridays.

Solution

Use the following run criteria for the job:

```
5th friday within month
```

Explanation

This solution runs the job on the 5th Friday of the month in months that contain 5 Fridays.

Scheduling a job relative to another criteria

Objective

Run a job on the first Friday after the 3rd Thursday of each month. Note that this will not be the 3rd Friday of the month if the month starts on a Friday.

Solution

Use the following run criteria for the job:

```
3rd thursday of this month plus 1 day
```

Explanation

This solution runs the job on the day after the 3rd Thursday of the month. This is the first Friday after the 3rd Thursday of the month.

Scheduling a job to run every week and adjusting it for holidays

Objective

Schedule a job to run every Monday, unless Monday is a holiday. If Monday is a holiday, then run the job on the previous workday.

Solution

Use the following run criteria for the job:

```
monday less 0 workdays
```

Explanation

The job always runs on Mondays that are workdays. When Monday is a holiday, and thus a non-workday, the job runs on the previous workday.

Variation

If, instead, you wanted to run the job on the first workday after the holiday, you can use the following run criteria for the job:

```
monday plus 0 workdays
```

Variation

If, instead, you did not want to run the job at all when Monday falls on a holiday, you can use the following run criteria for the job:

```
monday except holidays
```

Scheduling a job based on Saturday and the 1st day of the month

Objective

Run a job on the 1st Saturday of the month unless the 1st Saturday is also the 1st day of the month. If it is, then run the job on the 2nd Saturday of the month.

Solution

Use the following run criteria for the job:

```
saturday 2nd monthly
```

Explanation

Since this job does not run on the 1st day of the month, the earliest it can run is the 2nd day of the month. Thus, the job runs on the 1st Saturday on or after the 2nd day of the month, which translates to “saturday 2nd monthly”.

Scheduling a job based on the day of week for the 1st day of month

Objective

Run a job on the 1st Monday of each month. If the month starts on a Saturday or a Sunday, however, run the job on the 2nd Monday of the month instead.

Solution

To schedule a job based on the day of week for the 1st day of month

1. Use the following JavaScript script at Event trigger time for the job:

```
genTime('one','1st day of this month');
if (oneDAY=='Saturday' || oneDAY=='Sunday')
  WOB.RunABC='2nd monday of month';
else WOB.RunABC='1st monday of month';
```

2. Use the %WOB.RunABC variable as the run frequency in the job definition.

Explanation

The solution uses the genTime built-in function to generate date and time variables for the 1st day of the current month. In this example, these variables are prefixed with “one”.

The day of the week variable (oneDAY) is checked, and a variable representing the job's run frequency is set accordingly.

- If the 1st day of month was a Saturday or a Sunday, the WOB.RunABC variable is set to “2nd monday of month”.
- Otherwise, the WOB.RunABC variable is set to “1st monday of month”.

Scheduling a job based on day and day of month number

Objective

Run a job on the last Saturday of the month, provided this date falls on the 26th through to the 31st day of the month. Otherwise, run the job on the following Saturday (in other words, the first Saturday of the next month).

Solution

To schedule a job based on day and day of month number

1. Use the following JavaScript script at Event trigger time for the job:

```
WOB.runme=false;
genTime('lw','today less 1 week');
if (today('last saturday of month') && APPL._SDD > '25')
  WOB.runme=true;
if (today('first saturday of month') && lwDD < '26')
  WOB.runme=true;
```

2. Use the %WOB.runme variable as the run frequency in the job definition.

Explanation

This solution uses the genTime built-in function to generate date and time variables for one week ago. These variables are prefixed with “lw”.

The server selects the job for submission when the WOB.runme variable is true. This occurs on either one of two Saturdays:

- If today is the last Saturday of the month and the date is after the 25th.
- If today is the first Saturday of the month and the date of the previous Saturday, as indicated by the genTime variable lwDD, had a date earlier than the 26th. In this situation, the job would not have run the previous Saturday and must therefore run today.

Scheduling a job to run on the last five workdays of the month

Objective

Schedule a job to run only on the last 5 workdays of each month.

Solution

To schedule a job to run on the last five workdays of the month

1. Use the following JavaScript script at Event trigger time for the job:

```
genTime('last5wd','last workday of month less 4 workdays');
if (APPL._SDD >= last5wdDD && today('workday'))
  WOB.runme=true;
else WOB.runme=false;
```

2. Use the %WOB.runme variable as the run frequency in the job definition.

Explanation

The genTime built-in function generates time and date variables for the 5th last workday of the month. If the scheduled day is greater than or equal to the day of the month for the fifth last workday of the month, as indicated by the last5wdDD variable, and it is a workday, then the job is scheduled to run.

Variation

Another solution is to use the following run criteria for the job:

```
Run last workday of month
Run last workday of month less 1 workday
Run last workday of month less 2 workdays
Run last workday of month less 3 workdays
Run last workday of month less 4 workdays
```

Scheduling a job relative to a special day

Objective

A job needs to run on the first Saturday on or after a special day.

Solution

Use the following run criteria for the job:

```
1st saturday of special_day
```

Explanation

This solution selects the job to run on the first instance of Saturday on or after the special day name.

Scheduling a job based on Sunday and workdays

Objective

Schedule a job to run on the first Sunday of the month. If the following Monday is a holiday, then run the job on the Monday instead. If both Monday and Tuesday are holidays, then run the job on the Tuesday instead. In other words, the job runs on the day before the first workday after the first Sunday of the month.

Solution

To schedule a job based on Sunday and workdays

1. Use the following JavaScript script at Event trigger time for the job:

   ```
   WOB.genTime('1sun','1st sunday of this month plus 1 workday');
   ```

2. Use the following run criteria for the job:

   ```
   %1sunDATE less 1 day
   ```

Explanation

This solution uses the genTime built-in function to generate date and time variables for the first workday after the first Sunday of the month. In this example, the variables are prefixed with "1sun". The job is selected to run one day prior to this date.

Note: The genTime function is prefixed with WOB because one of the resulting variables, lsunDATE, is used outside of the JavaScript script.

Scheduling a job to run based on a different day's holiday status

Objective

Schedule a job to run every Saturday, unless the following Monday falls on a holiday. If the following Monday falls on a holiday, the job should not run.

Solution

To schedule a job to run based on a different day's holiday status

1. Use the following JavaScript script at Event trigger time for the job:

```
if (today('saturday') && daysTo('holiday') != 2)
  WOB.runme='true';
else WOB.runme='false';
```

2. Use the %WOB.runme variable as the run frequency in the job definition.

Explanation

The job is scheduled to run if it is a Saturday and the number of days to the next holiday is not equal to 2.

Note: It is assumed in this example that Sunday cannot be defined as a holiday.

Scheduling a job to run based on the last day of the previous month

Objective

Run a job on the 2nd workday of each month but only if the last day of the previous month was a workday.

Solution

1. Use the following JavaScript script at Event trigger time for the job:

```
WOB.sched=false;
genTime('LWM','last workday of month starting today less 1 month');
genTime('LDM','last day of month starting today less 1 month');
if (LWMDATE==LDMDATE)
  WOB.sched='2nd workday of month';
```

2. Use the %WOB.sched variable as the run criteria for the job.

Explanation

This solution uses the genTime built-in function to generate date and time variables for the last workday of the previous month and for the last day of the previous month. If these two dates are equal then the last day of the previous month was a workday, and the job is selected to run on the 2nd workday of the month.

More information:

Determining if a day in the past was a workday (see page 25)

Scheduling a job to run based on when another job was scheduled

Objective

Job A runs every Wednesday, unless Wednesday is a holiday. If Wednesday is a holiday, job A runs on Thursday.

Schedule job B to run every Thursday, except when job A runs on Thursday. When job A runs on Thursday, run job B on Friday.

Solution

To schedule a job to run based on when another job was scheduled

1. Use the following JavaScript script at Event trigger time for the two jobs:

```
WOB.runA='false';
WOB.runB='false';
genTime('W','wednesday starting today less 2 days');
genTime('W0',WDATE + ' plus 0 workdays');
// Determine if job A should run
if (today('wednesday') && !today('holiday')) WOB.runA='true';
if (today('thursday') && yesterday('holiday')) WOB.runA='true';
// Determine if job B should run
if (today('thursday') && WOB.runA=='false') WOB.runB='true';
if (today('friday') & WDATE != W0DATE) WOB.runB='true';
```

2. Use the %WOB.runA variable as the run frequency for job A.
3. Use the %WOB.runB variable as the run frequency for job B.

Explanation

The genTime built-in function is used twice in this solution. The first genTime generates date and time variables for Wednesday of the current week. The second genTime generates date and time variables for the first workday on or after Wednesday's date. If the two genTime dates are not equal, then Wednesday was a holiday.

Two variables, WOB.runA and WOB.runB, are used to represent the run frequencies of job A and job B respectively. Each of these variables is set to either true (select job), or false (do not select job).

Job A is selected to run when either of these conditions is true:

- Today is Wednesday and it is not a holiday.
- Today is Thursday and yesterday was a holiday.

Job B is selected to run when either of these conditions is true:

- Today is Thursday and job A is not selected to run.
- Today is Friday and Wednesday was a holiday.

More information:

Determining if a day in the past was a workday (see page 25)

Scheduling a job to run every two weeks

Objective

A job needs to run every two weeks and belongs to an Application with jobs with many different frequencies. The job was scheduled to run on March 13, 2006.

Solution

To schedule a job to run every two weeks

1. Use the following JavaScript script at Event trigger time for the job:

```
WOB.sched=false;
num=daysFrom('March 13,2006');
if (num%14==0) WOB.sched='today';
```

2. Use the %WOB.sched variable as the run criteria for the job.

Explanation

This solution uses the daysFrom built-in function to calculate the number of days since a prior run of the job (for example, March 13, 2006).

If this number is divisible by 14 (14 days is equivalent to two weeks) then a variable named WOB.sched is set to "today"; otherwise, it is set to false.

Variation

If all jobs in the Application need to run every two weeks, then you can simply schedule the Event every two weeks.

Scheduling a job to run every two weeks and advancing it for holidays

Objective

A job in an Application needs to run every two weeks on Friday. If this falls on a non-workday, run the job on the previous workday instead. Use Friday, August 1, 2003 as the starting point.

Solution

To schedule a job to run every two weeks and advance it for holidays

1. Use the following JavaScript script at Event trigger time for the job:

```
WOB.sched=false;
if (today('friday less 0 workdays'))
 {
  genTime('thisfri','friday');
  num=daysBetween('aug 1, 2003',thisfriDATE,'days');
  if (num%14==0) WOB.sched='today';
 }
```

2. Use the %WOB.sched variable as the run criteria for the job.

Explanation

The code handles two conditions: when Friday is a workday and when Friday is not a workday.

If today is Friday and it is a workday, the server generates date and time variables for today. A calculation is performed to see if the number of days from the starting point to today is a multiple of 14 (14 days is equivalent to two weeks). If it is, the WOB.sched variable is set to "today".

If Friday is a holiday and today is the last workday before Friday, the server generates date and time variables for this Friday. A calculation is performed to see if the number of days from the starting point to this coming Friday is a multiple of 14 (14 days is equivalent to two weeks). If it is, the WOB.sched variable is set to "today".

Scheduling a job on a random day of the month

Objective

A job needs to run at 11 a.m. on a random day each month.

Solution

To schedule a job on a random day of the month

1. Define a one-job Application. Use the following JavaScript script at Event trigger time for the job:

   ```
   genTime('nx','last day of month starting today plus 1 month');
   APPL.random_number=(Math.floor(Math.random()*nxDD)  +1).toFixed(0);
   ```

2. In the Do not submit before field for the job, enter **11am today plus %random_number days**.
3. Schedule an Event on the last day of the month to run your Application.

Explanation

The Application runs on the last day of each month and determines when the job should run in the following month. The genTime built-in function creates date and time variables, prefixed with nx, for the last day of the next month. The nxDD variable represents the number of days in the next month. This solution uses the Math.random function to generate a random number between 1 and nxDD, inclusive. The result is rounded and the result, with zero decimal places, is assigned to the APPL.random_number variable.

Note: Math.floor() rounds the result, and provides more evenly distributed results than Math.round().

The Application builds with the job delayed for the random number of days at 11 a.m.

Scheduling an Application to run within a date range

Objective

Run an Application at 4 p.m. each day between two dates, *date1* and *date2*, inclusively.

Solution

Use the following schedule criteria in the Event that runs the Application:

```
4pm daily starting date1 ending date2 plus 1 day
```

Alternatively, use the following schedule criteria:

```
4pm daily starting date1 ending 4:01pm date2
```

Explanation

By using “starting” and “ending” on the Schedule statement for the Event that runs the Application, the Event is scheduled from *date1* up to but not including *date2*. You can either add one day to the ending date or include a time on the ending date to ensure the schedule for *date2* is included.

For example, to run an Application at 4 p.m. each day between November 6, 2006 and April 22, 2007, use the following schedule criteria:

```
4pm daily starting Nov 6,2006 ending Apr 22,2007 plus 1 day
```

or

```
4pm daily starting Nov 6,2006 ending 4:01pm Apr 22,2007
```

After the last run, the Schedule statement is deleted but the Event remains in case you need to reschedule it or trigger it manually.

Scheduling a job to run within a date range

Objective

A job within an Application needs to run each Thursday between November 6, 2006 and April 22, 2007, inclusively.

Solution

To schedule a job to run within a date range

1. Use the following JavaScript script at Event trigger time for the job:

   ```
   WOB.sched=false;
   if (daysFrom('nov 6,2006') >= 0 && daysTo('apr 22,2007') >= 0)
     WOB.sched='thursday';
   ```

2. Use the %WOB.sched variable as the run criteria for the job.

Explanation

This solution uses the daysFrom and daysTo built-in functions to check if the scheduled date falls within the date range. If so, the job's run frequency is set to "Thursday". Otherwise, the job's run frequency is set to false.

Scheduling a monthly job with multiple criteria

Objective

Run a job on Sunday between the 10th and 16th day of each month except in February and August.

Solution

To schedule a monthly job with multiple criteria

1. Use the following JavaScript script at Event trigger time for the job:

```
WOB.runme='false';
if (today('sunday') && today('10th - 16th day of month') && !today('anyday of Feb Aug'))
  WOB.runme='true';
```

2. Use the %WOB.runme variable as the run frequency in the job definition.

Explanation

The job is selected to run if it is Sunday, it is between the 10th and 16th day of the month, and the month is not February or August.

Variation

Alternatively, you could use the following Run/Do not run conditions:

```
Run sunday
Do not run 1st-9th day of month
Do not run 17th-last day of month
Do not run anyday of feb aug
```

Variation

Since the job runs on the first Sunday after the 10th day of each month except in February and August, you could use the following:

```
Run sunday 10th monthly
Do not run anyday of feb aug
```

Alternatively, you could use:

```
Run sunday 10th monthly except anyday of feb and aug
```

Determining if a day in the past was a workday

Objective

Sometimes you may need to determine if some day in the past was a workday. For example, you may need to determine if last Friday was a workday, or if a fixed date fell on a workday, and schedule a job based on the result.

Solution

The following JavaScript script sets a variable, APPL.val, to either true or false based on whether October 13, 2005 was a workday.

```
genTime('one','oct 13,2005');
genTime('two',oneDATE + ' plus 0 workdays');
if (oneDATE == twoDATE) APPL.val='true';
  else APPL.val='false';
```

Explanation

This solution shows a general approach to checking if a day in the past was a workday. The first genTime generates date and time variables for the day in the past. This could be a criteria, such as “last day of month starting today less 1 month”, or it could be a date such as “oct 13, 2005”.

The second genTime generates date and time variables for the first workday on, or after, your first criteria. It uses the DATE variable from your first genTime and adds "plus 0 workdays".

The two dates from the genTimes are then compared. If they are equal then the day in the past was a workday; otherwise, it was a holiday.

More information:

Scheduling a job to run based on the last day of the previous month (see page 17)
Scheduling a job to run based on when another job was scheduled (see page 18)

Submitting a job at a particular time

Objective

An Application runs each day at 4 p.m., but one job within the Application must wait until 6 p.m.

Solution

Schedule an Event to run your Application at “4pm daily“. In the Do not submit before field for the job, specify a time dependency of “6pm“.

Explanation

You cannot use a time dependency as part of the run criteria. When an Application is generated, the run criteria determines whether a job should be selected to run. The Do not submit before field identifies a submission time for the job. This time can be an absolute time, such as “10pm”, or a time relative to the Event's scheduled or triggered time, such as “now plus 2 hours” or “realnow plus 3 hours”.

Running an Application multiple times a day

Objective

Each weekday, Monday through Friday, an Application needs to run multiple times a day. The scheduled times are 10 a.m., 2 p.m., 6 p.m., and 10 p.m.

Solution

When you define the Event to run the Application, use multiple schedule criteria such as the following:

```
Schedule 10am weekdays
Schedule 2pm weekdays
Schedule 6pm weekdays
Schedule 10pm weekdays
```

Explanation

This solution schedules the Event multiple times each weekday at the specified times.

Running a job multiple times in an Application

Objective

Within a daily Application containing many different jobs, there is one job that needs to run multiple times. Job MYJOB needs to run at 10 a.m., 2 p.m., 6 p.m., and 10 p.m.

Solution

To run a job multiple times in an Application

You can define the same job multiple times and use the Qualifier field to distinguish the jobs. Each instance of the job can have unique requirements, such as the time for submission.

One approach is to use the submission time as the job qualifier. For example, in the Qualifier field for the first instance of the job, you can use “10 AM”. The qualifiers and submit times for the different runs of the job are shown in the table below.

Job Name	Qualifier	Submit Time
MYJOB	10AM	10 AM
MYJOB	2PM	2 PM
MYJOB	6PM	6 PM
MYJOB	10PM	10 PM

In the Do not submit before field for each instance of the job, use the %WOB._qualifier built-in variable.

Explanation

When the Application generates, the server resolves the submission time for each job based on the job qualifier.

If you need to change the submission time for any of these jobs, change the job qualifier. For example, by renaming MYJOB.10AM (*jobname*.*qualifier*) to MYJOB.11AM, the job will now be scheduled to run at 11 a.m.

Scheduling an hourly Application within a time range

Objective

Schedule an Application to run every hour from 8 a.m. to 4 p.m. on workdays.

Solution

To schedule an hourly Application within a time range

1. Schedule an Event every hour to run your Application.
2. Use the Suspend at field to suspend the Event at 16:01 workdays.
3. Use the Resume at field to resume the Event at 07:59 workdays.

You can use the list scheduled Events feature to verify the correct scheduling of your Event.

Explanation

The Event is scheduled every hour on the hour. Suspend and resume criteria are used within the Event to control processing. The Event is suspended just after its 4 p.m. execution on workdays and resumed just prior to its 8 a.m. execution on workdays. This lets the Event to run hourly between 8 a.m. and 4 p.m. on workdays.

Variation

Another solution is to use multiple Schedule statements to schedule the Event multiple times each workday, for example:

```
Schedule 8am workdays
Schedule 9am workdays
.
.
.
 Schedule 4pm workdays
```

Scheduling a job to run based on day and time

Objective

An Event is scheduled every hour on the hour to run an Application. Job X in the Application should only be selected to run on Fridays at 3 p.m.

Solution

To schedule a job to run based on day and time

1. Use the following JavaScript script at Event trigger time for job X:

```
if (today('friday') && APPL._SHH=='15')
  WOB.runme='true';
else WOB.runme='false';
```

2. Use the %WOB.runme variable as the run frequency in the job definition.
3. Schedule an hourly Event to run the Application.

Explanation

Job X is selected to run only if it is Friday and the Event's scheduled hour (APPL._SHH) is 15 (in other words, the Event was scheduled at 3 p.m.).

Running different jobs based on the scheduled hour

Objective

An Event is scheduled every hour on the hour. Different jobs run based on the scheduled hour.

The following is the schedule frequency for each job:

- JOBA runs at 13:00, 15:00, and 17:00.
- JOBC runs at 08:00, 12:00, and 16:00.
- All other jobs run each hour.

Solution

To run different jobs based on the scheduled hour

1. Use the following JavaScript script at Event trigger time for each job:

```
WOB.runme=false;
//run different jobs based on scheduled hour
switch (WOB._name)
{
 case 'JOBA':
   if (APPL._SHH == '13' || APPL._SHH == '15' ||
     APPL._SHH == '17') WOB.runme=true;
   break;
 case 'JOBC':
   if (APPL._SHH == '08' || APPL._SHH == '12' ||
     APPL._SHH == '16') WOB.runme=true;
   break;
 default:
 WOB.runme=true;
}
```

2. Use the %WOB.runme variable as the run criteria for each job.

Explanation

The WOB.runme variable is set based on the job name and the scheduled hour. “case” statements are used for jobs with special requirements. If the name of a job in the Application does not match one of the case statement labels, WOB.runme is set to true for the job.

Scheduling a different job to run every hour

Objective

A one-job Application runs each hour. The name of the job in the Application depends upon its scheduled time. The job names are named MYJOB followed by a letter of the alphabet which is based on the scheduled hour. At 1 a.m. use the letter A, at 2 a.m. use the letter B, and so on, as shown in the following table:

Time	Job Name
1 a.m.	MYJOBA
2 a.m.	MYJOBB
3 a.m.	MYJOBC
...	...

Time	Job Name
11 p.m.	MYJOBW
12 a.m.	MYJOBX

Solution

To schedule a different job to run every hour

1. Schedule an Event hourly to submit the one-job Application.
2. Use the following JavaScript script at Event trigger time for the job:

```
Letters='ABCDEFGHIJKLMNOPQRSTUVWX';
if (APPL._SHH == '00') hour = '24';
else hour = APPL._SHH;
WOB.LastChar=Letters.substring(hour-1,hour);
```

3. For the name of the job, use the following:

```
MYJOB%LastChar
```

Explanation

The solution uses a symbolic variable named Letters, which is assigned a string representing the letters of the alphabet from A to X. Each hour, the server uses a different substring of this variable based on the scheduled hour (APPL._SHH).

If the scheduled hour is midnight, then the “hour” variable is assigned the value of 24 to enable the correct substring. Otherwise, “hour” is assigned the value of the APPL._SHH variable.

The LastChar variable is assigned the letter to be used based on the scheduled hour. For example, at 4 a.m., LastChar is determined by Letters.substring(3,4) which has the value D. Remember that the first position in a string is 0. Job MYJOBD is submitted.

You can also use %LastChar in the path to the Script name or pass this variable as an argument to the script being run.

Scheduling a different script each quarter

Objective

Schedule a job and run a different script for the job each calendar quarter. Each calendar quarter consists of three months, and the first quarter starts on January 1.

Solution

To schedule a different script each quarter

1. Use the following JavaScript script at run time for the job:

```
if (today('jan feb mar')) WOB.scriptname='quarter1';
if (today('apr may jun')) WOB.scriptname='quarter2';
if (today('jul aug sep')) WOB.scriptname='quarter3';
if (today('oct nov dec')) WOB.scriptname='quarter4';
```

2. Use the %WOB.scriptname variable in the Script/command name field for the job, for example:

```
/export/home/jsmith/%WOB.scriptname
```

Explanation

In a single job definition, the JavaScript script determines the current calendar quarter, and sets the WOB.scriptname variable to “quarter1”, “quarter2”, “quarter3”, or “quarter4”. When the job is ready to run, the server resolves this variable and runs the appropriate script.

Scheduling a cyclic Application

Objective

A group of jobs runs multiple times a day. The first run is at midnight. After all of the jobs have completed successfully, the next generation should run 60 minutes later. The last run each day should occur no later than 7 p.m.

Solution

To schedule a cyclic Application

1. Use a link at the end of your Application (in other words, as a successor to all other jobs) with a Delay submission when eligible by time of 60 minutes. Use the following JavaScript script at run time for the link:

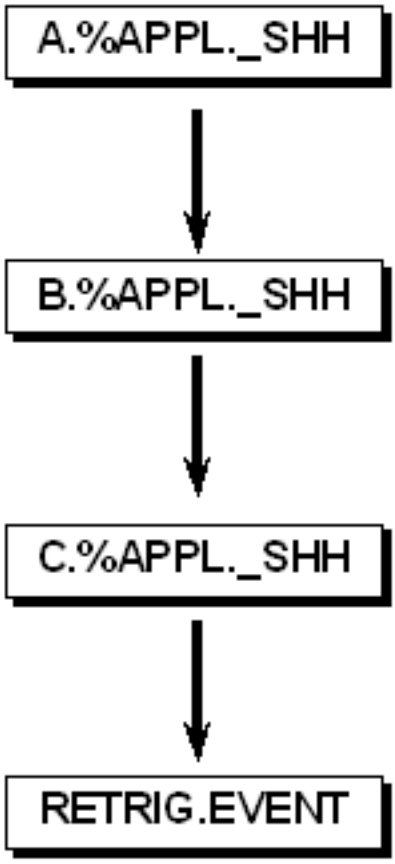
```
if (WOB._RHH < '19' && APPL._SDATE == WOB._RDATE)
  execTrigger('%APPL._event','ADD');
```

2. Schedule an Event at midnight each day to run the Application.

Explanation

The Event is scheduled at midnight each day to run the first group of jobs. Optionally, you can use time-based qualifiers for the jobs to distinguish each group of jobs for ease of monitoring.

The flow looks like this:



A link named RETRIG.EVENT waits for all jobs in the Application to complete successfully. This RETRIG.EVENT link waits 60 minutes before it becomes ready. It then checks the actual time (WOB._RHH) and if the time is before 7 p.m. on the same day, it retriggers the original Event. The execTrigger built-in function with the ADD option is used so that the regular daily scheduled Event at midnight is not affected.

Keeping an Application active

Objective

There are a number of standalone ad hoc jobs that can be requested by a user. When a user requests you to run one of these jobs, you want to run the job by inserting the job into an active Application.

Solution

To keep an Application active

You can schedule an Application each day to handle ad hoc jobs. In the Application, define a link with a Do not submit before time that is the usual cutoff time for receiving requests (for example, 8pm). This time dependency keeps the Application active until the time you specify and allows you to insert jobs.

When you want to run an ad hoc job, insert the job into this Application.

Explanation

An Application needs to be active for you to insert jobs. The KEEPOPEN link keeps the Application active until the Do not submit before time you specify. The Application continues to stay active past the submit time for the link until all inserted jobs complete successfully.

Running ad hoc jobs

Objective

A number of independent ad hoc jobs are defined in an Application. When a request comes in to run one of these jobs, there needs to be an easy way of running the job.

Solution

To run ad hoc jobs

1. Define each of your ad hoc jobs in the Application.
2. Use Run Daily or Run Anyday as the run criteria for each job. You can set this as a job default in your Application by using the Run frequency Application property.
3. Define an Event to run your Application. This Event will be triggered manually.
4. To run one of these ad hoc jobs, trigger the Event and specify the job name you want to run in the Root jobs to run field. For example, to run job Myjob123, specify Myjob123 in the Root jobs to run field.

   Note: The job name is not case sensitive. You could specify MYJOB123, myjob123, or MyJoB123.

Explanation

When you specify a single job name in the Root jobs to run field, only that job is selected to run. This lets you predefine ad hoc jobs and their requirements, and run them on demand.

In the Root jobs to run field, you can also specify a job name followed by a plus (+) sign to include the job and all its successors, or specify a list that contains job names and/or job names followed by a plus sign. For example, specify Myjob123, Myjob456 to run both of these ad hoc jobs.

Note: If an ad hoc job has predecessors and successors, then normally you would define this as an on-request job in the Application containing the related jobs. You can request such jobs from Monitor using the Request command.

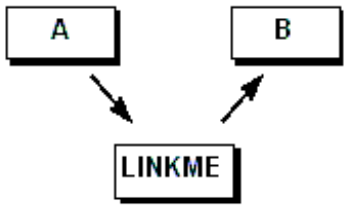
Adding a relationship between two unrelated jobs

Objective

In an Application, two jobs (A and B) are unrelated. For the current run only, you want to run job B after job A.

Solution

Prior to job B running, you insert a link with a predecessor of A and a successor of B to "link" the two jobs.



Explanation

The server automatically completes a link as soon as its dependencies are met. When job A completes, the link becomes ready, completes itself, and releases job B.

Providing status for a long-running Application

Objective

A long-running Application is scheduled to run at 8 a.m. each day. If the Application is not complete by 8 a.m. the next day, send a message to indicate the Application is still processing.

Solution

To provide notification for a long-running Application

1. Use a link with a Do not submit before time of 8am tomorrow. The link has no job dependencies.
2. Customize the email message to send when the link reaches the COMPLETE state; for example, %APPL._name is still processing from %APPL._SDATE. This message uses built-in variables for the name of the Application and the scheduled date.
3. Define the link as a conditional job because it may not process.

Explanation

The link, which has no dependencies other than time, sends a message at 8 a.m. the next day if the Application is still processing.

There are two possible results:

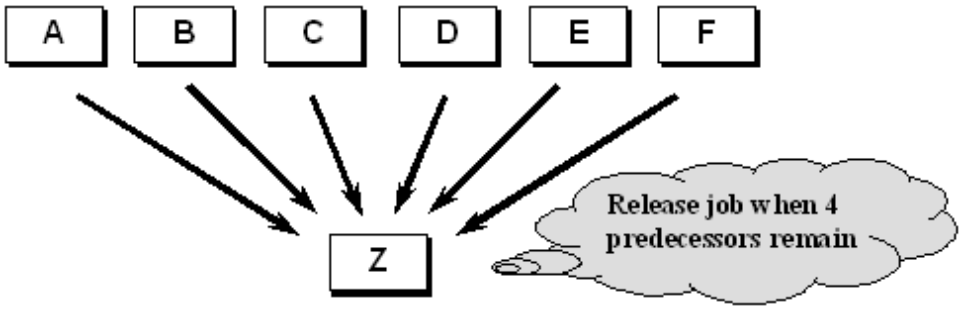
- If all other jobs in the Application are complete prior to 8 a.m. the next day, the link is bypassed (since it is a conditional job) and the Application completes.
- Otherwise, if jobs are still processing at 8 a.m. the next day, a message is sent at that time.

Setting up a job with optional predecessors

Objective

There are six jobs (A, B, C, D, E, and F), and you need to run another job (Z) after any two of them complete successfully.

The requirement looks like this:



Solution

To set up optional predecessors

1. Define each of the six predecessor jobs as conditional jobs so that the Application can be considered complete without all of the jobs completing.
2. Set up each of the six jobs as a predecessor to job Z.
3. In the General dialog for job Z, enter a release count of 4 in the Release job when field.

Explanation

By default, the release count for a job is 0, which means that each of its predecessors needs to complete before it is released. In situations where a job has optional predecessors, you can indicate a release count.

Job Z initially has a hold count of 6 because it has 6 predecessors. Job Z is released when its hold count reaches 4. This is the case when 2 of its predecessors complete successfully.

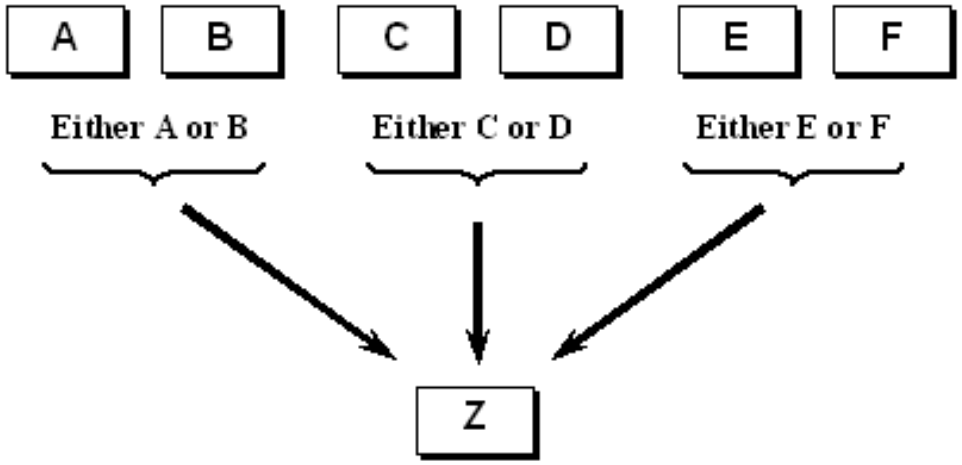
After job Z completes, some of its predecessors may not have completed. Since you have defined the predecessors as conditional jobs, when job Z completes, the server bypasses any conditional jobs that are not complete and completes the Application.

Setting up groups of optional predecessors

Objective

There are three groups of jobs with two jobs in each group. One job from each group must complete before running a successor job.

The dependencies look like this:

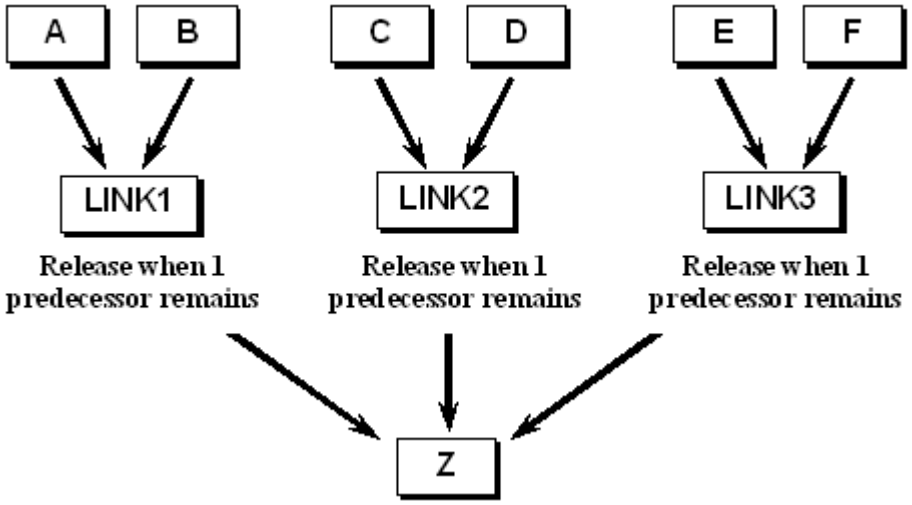


Solution

To set up groups of optional predecessors

1. Set up a link as a successor for each group of jobs.
2. Specify 1 in the Release job when field for each of these Links.
3. Set up each of the three Links to release successor job Z.

The solution looks like this:



Explanation

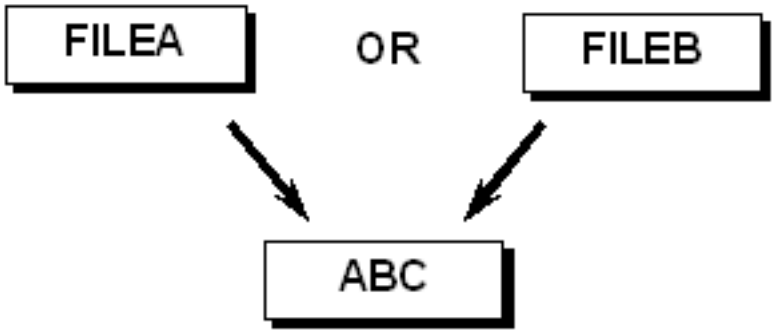
Initially, job Z has a hold count of 3 because it is waiting for each of the Links to complete. Each link completes when one of its predecessors completes successfully. In other words, when one job in each of the three groups completes successfully, the Links complete, and the successor job runs.

Setting up optional file trigger dependencies

Objective

Job ABC cannot run until one of two files, either “FILEA” or “FILEB”, has been created. Whichever one is created first should cause the server to submit the job. Also, the Application does not need to wait for the other file before it completes.

The dependencies look like this:



Solution

To set up optional file trigger dependencies

1. Use a file trigger job, FILEA and FILEB, for each file dependency, and define the file trigger conditions (for example, trigger upon creation). Set up each of these jobs as a predecessor to job ABC.
2. In the Release job when field for job ABC, specify 1 to indicate it should be released when 1 predecessor completes.
3. Define each of the file triggers as conditional jobs (Conditionally) so that the Application can be considered complete without both of them completing.

Explanation

When the server builds the Application, job ABC has a hold count of 2. By specifying “Release Job When 1 predecessor remains” for job ABC, the server releases this job when its hold count becomes 1. This means the server releases job ABC when either FILEA or FILEB completes (in other words, when the file name specified in its job definition is created).

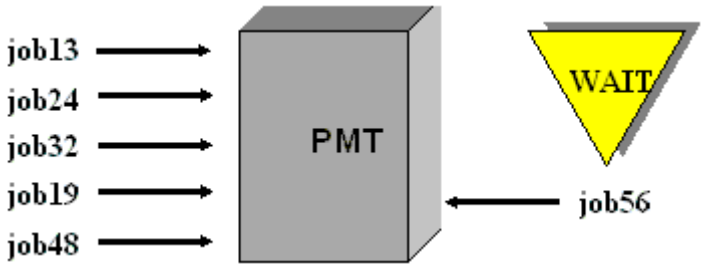
The “Submit conditionally” setting on the FILEA and FILEB definitions indicates that these jobs do not have to complete for the Application to complete. For example, if FILEA completes and job ABC runs to successful completion, the server bypasses FILEB and the Application completes.

Controlling concurrent access to a system

Objective

There are five connections allowed to a system referred to as the PMT system. If more than five jobs try to access the system at the same time, the jobs fail and need to be restarted. There is a need to limit the number of jobs that access the system at any time to five.

The requirement looks like this:



Solution

To control the maximum number of concurrent job executions

1. In the Services perspective, define a renewable resource named PMT with both Availability and Maximum availability set to 5.
2. For each job that requires access to the PMT system, specify that the resource requires 1 unit of the resource in the job definition Resources dialog.

Explanation

Since there are only 5 units of this renewable resource defined, only five of these jobs can execute at a time.

If five jobs are using the resource, then any other jobs needing the resource go into a RESWAIT state. As each job completes (successfully or not), one unit of the resource is freed up.

Running mutually exclusive groups of jobs

Objective

One group of jobs cannot execute while another group of jobs is running.

The following are the criteria for this example:

- One group of jobs consists of jobs B and C.
- The other group of jobs consists of jobs D and E.
- All jobs in one group must complete successfully before processing any jobs in the other group.

Solution

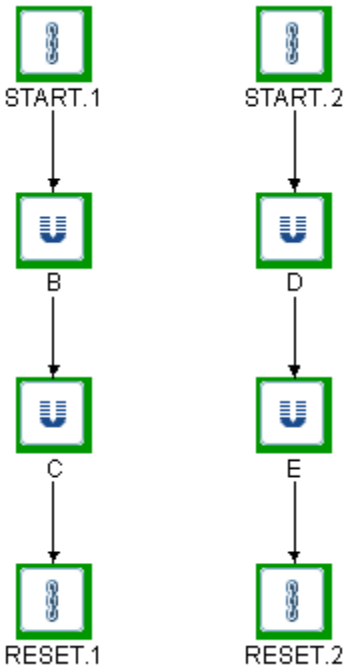
To run mutually exclusive groups of jobs

The solution involves using a depletable resource that depletes to zero when one sequence starts and replenishes when that sequence ends.

1. In the Services perspective, define a depletable resource named ONESTREAM with Availability set to 1.
2. Use a different link as a predecessor to each group of jobs. Each link requires one unit of the depletable resource and completes itself when the resource requirement is met.
3. Use a link at the end of each group of jobs that uses the following JavaScript script at run time to reset the available count of the depletable resource to 1.

```
resetResourceProperty('ONESTREAM','Availability','1');
```

The solution looks like this:



Explanation

In this example, the server controls resources such that one group of jobs does not execute while another group is running. The technique illustrated here can be used even if both groups of jobs are in different Applications.

When one stream of jobs starts, the link at the beginning of the stream (START.1 or START.2) uses the one available unit of the depletable resource and thus prevents the other stream from starting. When one stream completes, a link (RESET.1 or RESET.2) resets the resource count to 1 to let the other stream run.

The depletable resource is a requirement for each link instead of a requirement for the first job in each path. This prevents a problem that could occur if the first job fails and cannot be restarted because the resource isn't available.

Using a renewable resource with an Availability count of 1 is not sufficient here as you want to ensure that once one path starts, it goes to completion. Also, you may not know which path will start first.

Planning for system shutdown

Objective

When a particular system is due to be shut down, long-running jobs should be prevented from starting and notification needs to be provided.

Solution

To plan for a system shutdown

1. In the Services perspective, define a threshold resource named ELAPSED representing elapsed time in minutes.

   Initially, you can assign ELAPSED a Maximum availability count such as 1440 to represent 24 hours.

2. Specify the ELAPSED resource as a dependency for the long-running jobs you want to prevent from starting prior to system shutdown.

   For example, if a job usually runs for 2 hours, specify the job requires 120 units of the ELAPSED resource.

3. Set up an Application with a countdown procedure that you can use prior to a system shutdown. You can use a link that runs the following JavaScript script at Event trigger time to set the available count of the resource and then retrigger itself one minute later:

   ```
   time=APPL._user1;
   resetResourceProperty('ELAPSED','Availability',time);
   APPL.remain=(time - 1).toFixed(0);
   if (APPL.remain >= 0)
     execTrigger('%APPL._event','','now plus 1 minute','','%APPL.remain');
   ```

4. Indicate that an SNMP trap should be sent when the link becomes ready. Customize the message to indicate the number of minutes until shutdown, for example:

   ```
   System shutdown in %remain minutes
   ```

5. Define an Event (for example, OPER.SHUTDOWN) that runs your Application. When you trigger this Event, specify the number of minutes until shutdown in the Parameter 1 field.

Explanation

When you trigger the Event, you pass the number of minutes until shutdown.

The Application run by the Event contains a link that does the following:

- Assigns the APPL._user1 variable to a variable named “time”.
- Sets the available count of a threshold resource named ELAPSED to the number of minutes until shutdown.
- Decrements the time variable by 1.
- If the amount of time now left until shutdown is greater than or equal to zero, the server retriggers the Event in one minute. The amount of time left is passed as user parameter 1.
- This recursive procedure repeats until the amount of time left is less than zero.

Jobs requiring the ELAPSED resource will not run unless a sufficient quantity of the resource is available.

Using job completion as a resource

Objective

This example uses a threshold resource to represent the completion of a job. For example, you can have a job in one Application that is dependent on an on-request job with an unpredictable frequency in another Application.

In this example, PAYJOB2 requires that the previous run of PAYJOB1 has completed successfully.

Solution

To schedule a job dependent on an on-request job

1. In the Services perspective, define a threshold resource named PAYJOB1 with Availability of 0 to represent the successful completion of PAYJOB1. In this example, the name of the resource is the same as the name of the predecessor job, PAYJOB1.
2. For the successor job, define the first job as a resource requirement. In other words, PAYJOB2 requires 1 unit of the PAYJOB1 resource.
3. Use a link in each Application to set the available count of the resource.

   When PAYJOB1 completes successfully, use a link to set the available count of the PAYJOB1 resource to 1. Use the following JavaScript script:

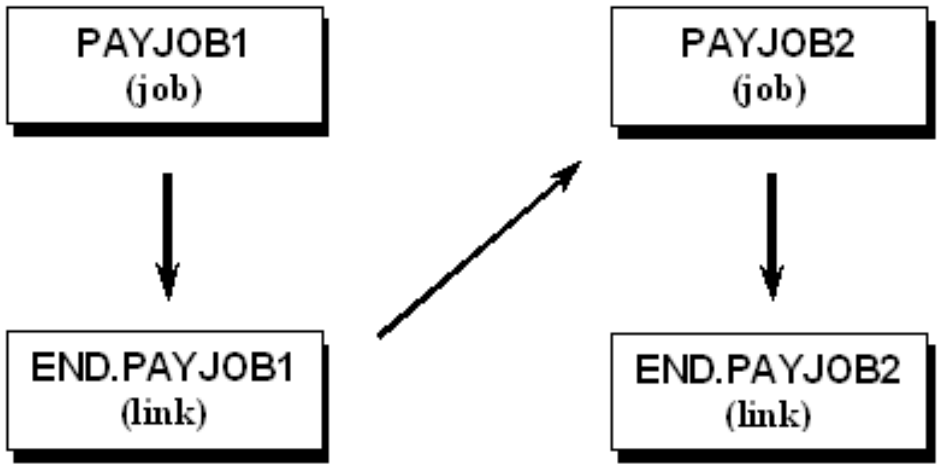
   ```
   resetResourceProperty('PAYJOB1','Availability','1');
   ```

   When PAYJOB2 completes successfully, use a link to set the available count of the PAYJOB1 resource to 0. Use the following JavaScript script at run time for this link:

   ```
   resetResourceProperty('PAYJOB1','Availability','0');
   ```

Explanation

The dependencies look like this:



This solution uses a threshold resource to represent the completion of job PAYJOB1.

The server produces the following results:

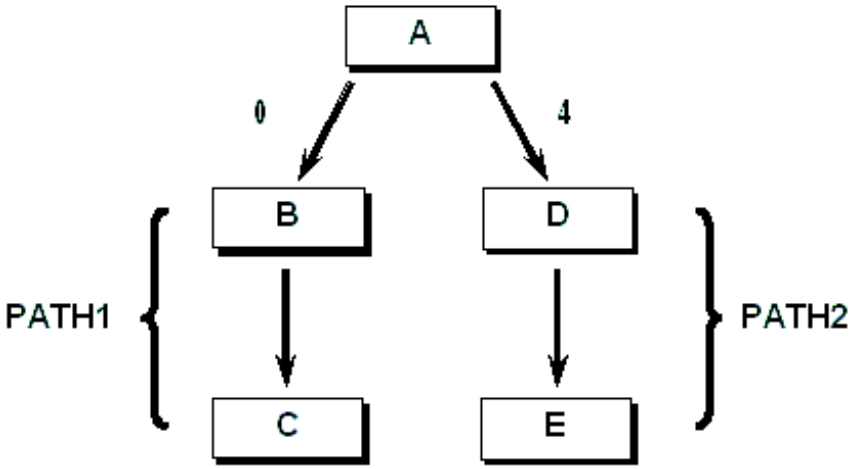
- When PAYJOB1 ends successfully, the server runs a link named END.PAYJOB1. The link sets the available count of the PAYJOB1 resource to 1.
- The server does not submit PAYJOB2 until the PAYJOB1 resource is available.
- When PAYJOB2 completes successfully, the server runs a link named END.PAYJOB2. The link sets the available count of the PAYJOB1 resource to 0. This prevents PAYJOB2 from running without its dependency the next time it is scheduled.

Running different jobs based on the exit code of predecessor

Objective

A job can end successfully with either an exit code of 0 or an exit code of 4. Different groups, or paths, of successor jobs need to run based on a successful exit code.

The dependencies look like this:



Solution

The server can release a job based on the exit code of a predecessor. For example, job A can release job B on an exit code of 0 and release job D on an exit code of 4.

To run different jobs based on the exit code of predecessor

You specify these release conditions when you define the jobs in the Define perspective.

1. To specify a release condition, right-click the dependency line between the two jobs, and select Release conditions.
2. Set up job A to release job B on an exit code of 0; set up job A to release job D on an exit code of 4.
3. Set up job B to release job C; set up job D to release job E. By default, a job releases its successor based on a successful completion.
4. Identify 4 as a successful exit code for job A. Note that it isn't necessary to identify 0 as a successful exit code for job A, as this is the default.

Explanation

This example shows two success paths. Since a job can end with only one exit code, only one path is eligible to run.

If job A completes with an exit code of 0:

- Jobs B and C run.
- Jobs D and E are bypassed.

If Job A completes with an exit code of 4:

- Jobs D and E run.
- Jobs B and C are bypassed.

This approach only works when there are different success paths. As long as job A is successful, only one of the two paths will run. The server automatically bypasses the jobs in the success path that do not run.

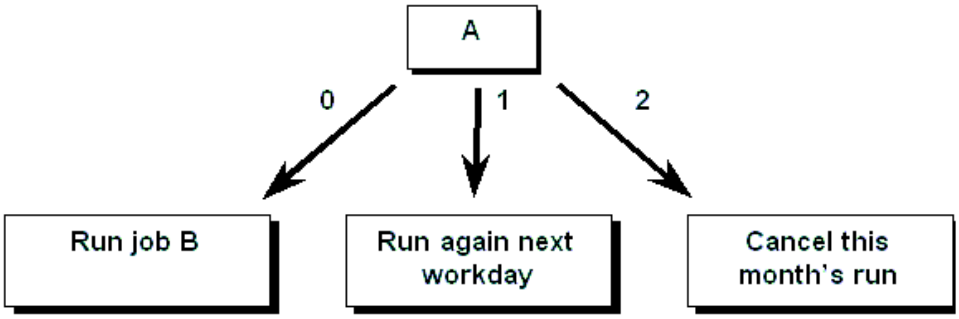
Taking different actions based on 1 of 3 successful return codes

Objective

A monthly Application is scheduled for 10 a.m. on the last workday of each month. The first job, job A, can end successfully with an exit code of 0, 1, or 2. Different actions are required based on the exit code, as described in the following table.

Return Code for Job A	Action
0	Run job B
1	Complete Application and run it again at 10 a.m. the next workday
2	Cancel this month's run

The requirements look like this:

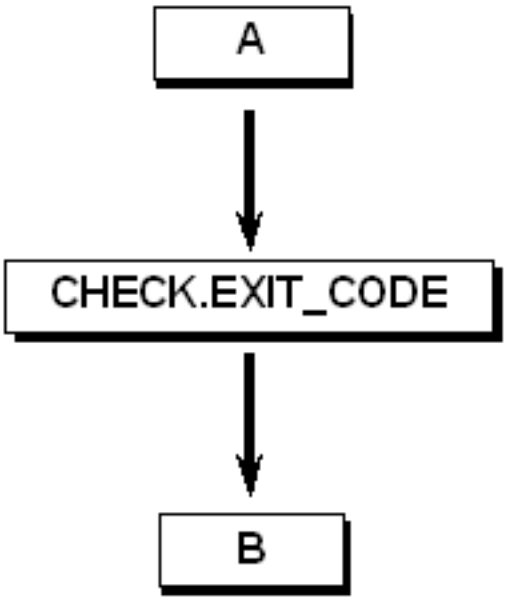


Solution

To take different actions based on 1 of 3 successful return codes

1. When you set up the Application, set up job A to release a task (for example, CHECK.EXIT_CODE), which then releases job B.

   The dependencies look like this:



2. Use the Exit Codes dialog for job A to indicate that 1 and 2 are successful exit codes. Note that 0 is a successful exit code by default.

3. Use an Alert for job A when it reaches a Complete state, which runs the following JavaScript script:

```
switch (WOB._cmpc)

{
case '0':
// if cmpc is 0 then complete task and let job b run
 execCommand('check.exit_code','%(APPL._name).%APPL._gen',
 'ACTION COMPLETE');
 break;
case '1':
// if cmpc is 1 then complete appl and trigger next workday
 execCommand('ALL','%(APPL._name).%APPL._gen',
 'ACTION COMPLETE');
 execTrigger('%APPL._event','ADD','10AM today plus 1 workday');
 break;
case '2':
// if cmpc is 2 then complete appl
execCommand('ALL','%(APPL._name).%APPL._gen',
 'ACTION COMPLETE');
}
```

4. Schedule an Event at 10 a.m. on the last workday of each month to run your Application.

Explanation

When the first job, job A, completes successfully, an Alert triggers. The Alert's JavaScript script checks the exit code for job A (WOB._cmpc) and takes the appropriate action. This solution uses “case” statements.

A task, CHECK.EXIT_CODE, is set up as a successor to job A. This prevents job B from running until the server can check the exit code of job A.

Built-in variables are used for the Event name, Application name, and Application generation number.

Variation

Alternatively, you can use the following JavaScript script. This solution uses “if” statements instead of “case” statements.

```
// if cmpc is 0 then complete task and let job b run
if (WOB._cmpc==0)
 execCommand('check_exitcode_a','%(APPL._name).%APPL._gen',
  'ACTION COMPLETE');
// if cmpc is 1 then complete appl and trigger next workday
if (WOB._cmpc==1)
 {
 execCommand('ALL','%(APPL._name).%APPL._gen',
  'ACTION COMPLETE');
 execTrigger('%APPL._event','ADD','10AM today plus 1 workday');
 }
// if cmpc is 2 then complete appl
if (WOB._cmpc==2)
 execCommand('ALL','%(APPL._name).%APPL._gen','ACTION COMPLETE');
```

Resubmitting a job five minutes after it fails

Objective

If a particular job fails, it should be resubmitted after a five-minute delay.

Solution

To resubmit a job five minutes after it fails

1. When you define the job, use the Notifications dialog and identify an Alert to be triggered if the job fails.
2. Define the Alert, and use the following JavaScript script for the Alert:

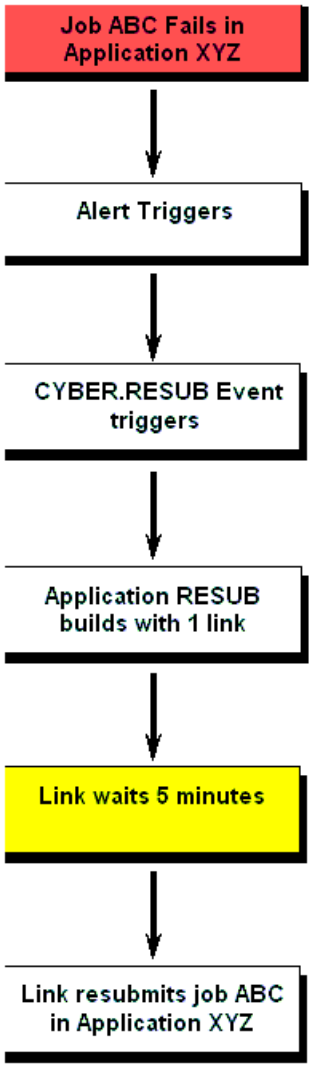
   ```
   execTrigger('CYBER.RESUB','','','','%WOB._name','%APPL._name','%APPL._gen');
   ```
3. Define a one-job Application containing a link.
   - Use a Do not submit before time of “now plus 5 minutes”.
   - Use the following JavaScript script at run time to resubmit the failed job.

     ```
     execCommand('%APPL._user1','%(APPL._user2).%APPL._user3','ACTION RESUB');
     ```
4. Define the CYBER.RESUB Event to run the one-job Application.

Explanation

If the job fails, an Alert triggers, which in turn triggers the CYBER.RESUB Event. The built-in variables for job name, Application name, and generation number are passed as user parameters to this Event. The Event builds an Application containing a link. The link waits for five minutes, and then uses the execCommand built-in function to resubmit the failed job.

The following diagram shows the flow from the time of job failure to the time of job resubmission.



Built-in variables are used to ensure the job is resubmitted in the correct generation of the correct Application. This lets you reuse the same technique for any number of jobs regardless of the Applications to which they belong.

Running a recovery job after failure

Objective

There are a number of similar jobs. When one of these jobs fails, you want to run a generic recovery Application that runs a recovery job and then completes the failed job.

Solution

To run a recovery job after failure

1. Indicate an Alert is to be triggered when one of these jobs fails.
2. Define the Alert. Use the following JavaScript script. This script uses the execTrigger built-in function to trigger a recovery Event, and passes the name of the failed job, the Application to which it belongs, and the generation number for that Application as user parameters, for example:

   ```
   execTrigger('cyber.recover','','','','%WOB._name','%APPL._name','%APPL._gen')
   ;
   ```

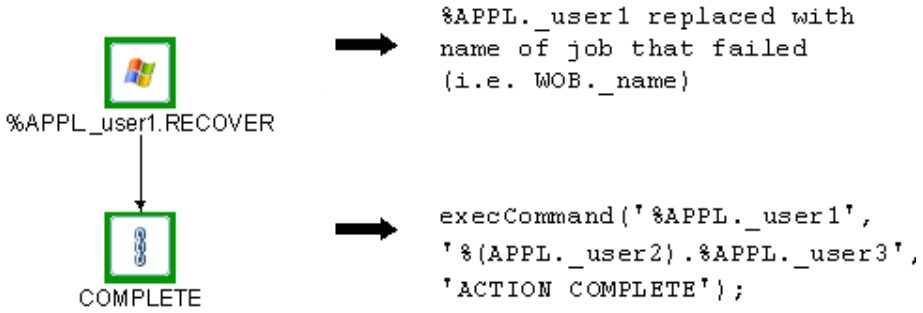
3. Set up the recovery Application as follows:

   a. Define the recovery job. For the name of the job, you can use the name of the failed job followed by a qualifier of RECOVER (for example, %APPL._user1.RECOVER).

   b. Set up the recovery job to release a link. For the link, run a JavaScript script at run time that uses the execCommand built-in function to complete the failed job, for example:

   ```
   execCommand('%APPL._user1','%(APPL._user2).%APPL._user3','ACTION
   COMPLETE');
   ```

   The recovery Application looks like this:



4. Define an Event (CYBER.RECOVER) to run the recovery Application. This Event will be triggered automatically.

Explanation

When the job fails, an Alert builds a recovery Application. This Application consists of a recovery job followed by a link that completes the failed job in the original Application.

Built-in variables for the job name, Application name, and generation number are passed to the recovery Application as user parameters. This lets you use the same generic Application for jobs requiring a similar recovery process.

Delaying job submission until the next hour

Objective

When a job's predecessor is complete, the job needs to be delayed until the beginning of the next hour. For example, if job A completes at 10:17 a.m. then job B should be submitted at 11 a.m.

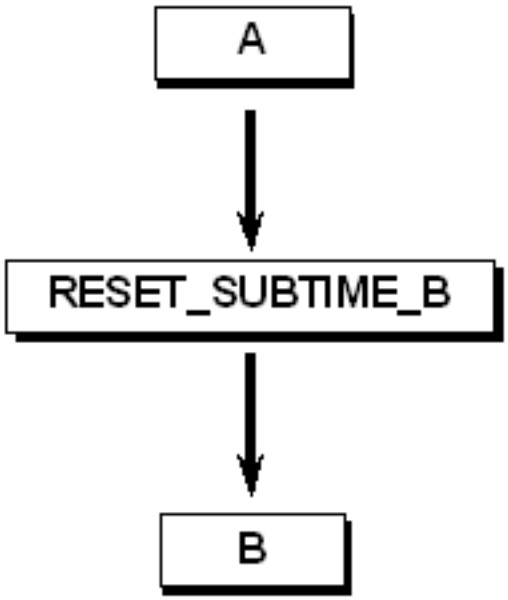
Solution

To delay job submission until the next hour

1. Set up a task (for example, RESET_SUBTIME_B) as a successor to job A and as a predecessor to job B.
2. Use the following JavaScript script at run time for the task:

```
execCommand('B','%(APPL._name).%APPL._gen','ACTION RESET EARLYSUB("HOURLY")');
execCommand('%WOB._name','%(APPL._name).%APPL._gen','ACTION COMPLETE');
```

The solution looks like this:



Explanation

Often you may need a job in an Application to wait for a particular time before submission. You can use the Do not submit before field to specify a delayed submission time, such as “7pm” or “now plus 30 minutes”. You can also use the Delay submission when eligible by field to delay submission of a job relative to the time the job's predecessors are all complete (for example, “30” delays the job by 30 minutes).

In this example, you do not know what time the job's predecessor will be complete, and so it is not possible to use either the Do not submit before field or the Delay submission when eligible by field.

When job A completes successfully, the task becomes ready. The task runs a JavaScript script that uses the execCommand built-in function to reset job B's submit time to the beginning of the next hour, and then completes itself.

Built-in variables are used on each execCommand to represent the name of the Application (%APPL._name) and the generation number (%APPL._gen) to ensure the actions are performed against the correct generation of the Application.

The second execCommand uses the built-in variable for the job name (%WOB._name). This is a common practice for self-completing Tasks. It lets you reuse the code in other Applications and lets you change the name of the task without any changes to the JavaScript script.

Bypassing a job without waiting for its predecessor

Objective

Jobs A, B, and C are three sequential, daily jobs. If job B has not been submitted by 10 a.m., job C should start (even if job A is running), and job B should not run.

Solution

To bypass a job without waiting for its predecessor

1. Set up the three sequential, daily jobs in an Application.
2. For job B, in the Submission field of the Time Dependencies dialog, specify a time of 10AM.
3. Use an Alert for job B when it reaches an Abandon Submission state.
4. Use the following JavaScript script for the Alert:

```
execCommand('%WOB._name','%(APPL._name).%APPL._gen','ACTION COMPLETE');
```

Explanation

When a job's abandon submission time is met, the job is flagged to be bypassed. The job is not actually bypassed until its predecessors are complete. In this scenario, job C can start once it is determined that job B will be bypassed (in other words, when job B's abandon submission time is met).

If job B is not submitted by 10 a.m., the job's submission is abandoned. The Alert triggers to force complete job B, and job C is submitted.

Built-in variables are used for the job name (%WOB._name), Application name (%APPL._name), and generation number (%APPL._gen) to ensure the correct job is marked complete in the correct generation of the Application.

Variation

Since job B usually waits for job A to complete successfully prior to being bypassed, another approach is to use the Predecessor dependencies field for job B to specify a time just after 10 a.m. (for example, 10:01 AM). If job B's abandon submission time is met, job B is flagged to be bypassed. Its dependencies are then dropped, causing it to be bypassed, and job C is eligible to start.

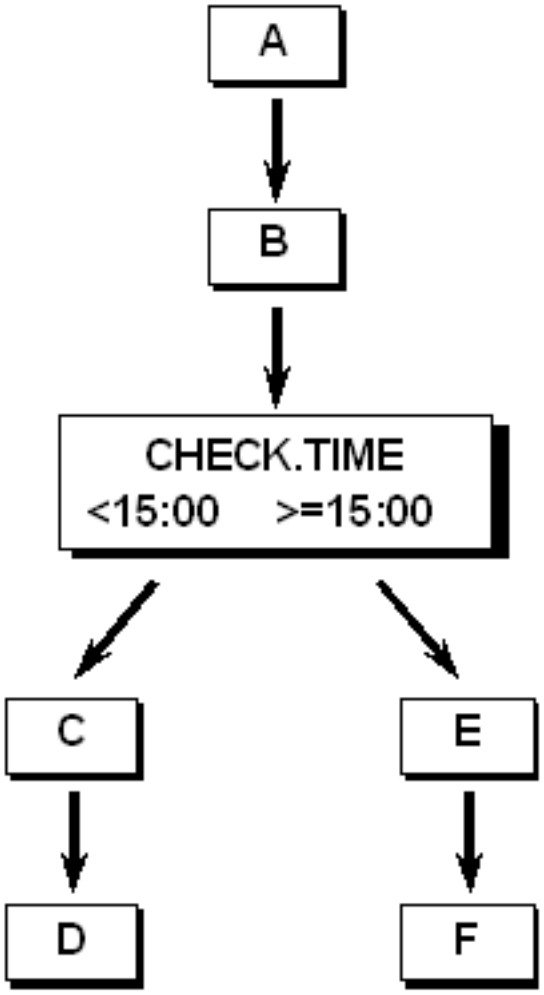
Running different jobs based on the time a predecessor job completes

Objective

After jobs A and B complete successfully, a different group of jobs needs to run based on what time it is. If the time that job B completes successfully is prior to 15:00, the server should run jobs C and D. If the time is 15:00 or later, the server should run jobs E and F.

Solution

The dependencies look like this:



To run different jobs based on the time a predecessor job completes

1. Define jobs C, D, E, and F as request jobs.

2. Use a task after job B that uses the following JavaScript script at run time. This script checks the time and takes the appropriate action.

```
if (WOB._RHH < '15')
 {
  execCommand('C','%(APPL._name).%APPL._gen','ACTION REQUEST');
  execCommand('D','%(APPL._name).%APPL._gen','ACTION REQUEST');
 }
else
 {
  execCommand('E','%(APPL._name).%APPL._gen','ACTION REQUEST');
  execCommand('F','%(APPL._name).%APPL._gen','ACTION REQUEST');
 }
execCommand('CHECK.TIME','%(APPL._name).%APPL._gen','ACTION COMPLETE');
```

Explanation

After job B completes successfully, the server uses a task (for example, CHECK.TIME) that runs a JavaScript script to check the actual time. If the time is prior to 15:00, the server requests jobs C and D. If the time is 15:00 or later, the server requests jobs E and F. In either case, the server completes the task to enable the successors to run. The request jobs that are not requested are automatically bypassed.

A task is used in this example so that the server can check the time and take action before all of the successor jobs are released.

Built-in variables are used on each execCommand to represent the name of the Application and the generation number to ensure the actions are performed against the correct generation of the Application.

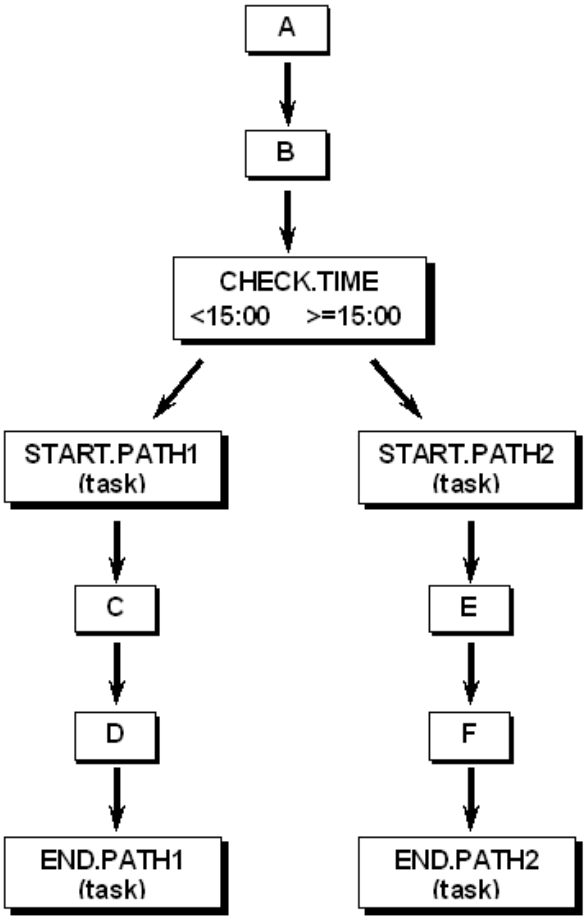
Running jobs in a different order based on time

Objective

After jobs A and B complete successfully, the order in which the successor jobs run is based on what time it is. If the time that job B completes successfully is prior to 15:00, the server should run jobs C and D, and then jobs E and F. If the time is 15:00 or later, the server should run jobs E and F, and then jobs C and D.

Solution

The solution looks like this:



To run jobs in a different order based on time

1. Add a task at the beginning and at the end of each of the two groups of successor jobs. The names in this example are START.PATH1, START. PATH2, END.PATH1, and END.PATH2.
2. Define END.PATH1 and END.PATH2 as request jobs.

3. Use a link after job B and before each of the START tasks that uses the following JavaScript script at run time. This script checks the time and takes the appropriate action.

```
//Determine which path to run first
if (WOB._RHH < '15')
 {
  execCommand('START.PATH1','%(APPL._name).%APPL._gen','ACTION COMPLETE');
  execCommand('END.PATH1','%(APPL._name).%APPL._gen','ACTION REQUEST');
 }
else
 {
  execCommand('START.PATH2','%(APPL._name).%APPL._gen','ACTION COMPLETE');
  execCommand('END.PATH2','%(APPL._name).%APPL._gen','ACTION REQUEST');
 }
```

4. Use the following JavaScript script at run time for END.PATH1. It completes the START.PATH2 task.

```
execCommand('START.PATH2','%(APPL._name).%APPL._gen','ACTION COMPLETE');
```

5. Use the following JavaScript script at run time for END.PATH2. It completes the START.PATH1 task.

```
execCommand('START.PATH1','%(APPL._name).%APPL._gen','ACTION COMPLETE');
```

Explanation

After job B completes successfully, the server uses a link (for example, CHECK.TIME) that runs a JavaScript script to check the actual time. If the time is prior to 15:00, the server completes START.PATH1 and requests END.PATH1. If the time is 15:00 or later, the server completes START.PATH2 and requests END.PATH2. When one group of jobs completes, the requested END task completes the task, which starts the other group of jobs.

Built-in variables are used on each execCommand to represent the name of the Application and the generation number to ensure the actions are performed against the correct generation of the Application.

Running the next scheduled Application early

Objective

You need to run the next day's version of an Application today.

Solution

As long as you need to replace the next scheduled instance of the Application, you can simply trigger the Event to run the Application and select Replace next scheduled Event.

For example, if an Application runs daily at 8 a.m., and you want to run Friday's Application on Thursday afternoon, you can trigger the Event now and use the Replace next scheduled Event option.

Explanation

The server automatically selects the jobs and resolves variables based on the “replaced” scheduled date.

More information:

Running an Application for any future date (see page 61)

Running an Application for any future date

Objective

You need to run a future version of an Application today.

Solution

Define another Event that runs the Application. Schedule the Event for the particular date in the future and use the “ONCE” keyword on the Schedule statement.

Trigger this Event and select Replace next scheduled Event.

Explanation

For example, if you want to run an Event today as if it was October 17, 2006, take the following steps:

1. Schedule another Event to run the Application and use “schedule oct 17, 2006 once”.
2. Trigger the Event with the Replace next scheduled Event option. The server triggers the Event now and replaces the October 17, 2006 execution.

The server selects jobs and resolves criteria variables based on this October 17, 2006 date.

The use of “ONCE” ensures the Event will not be scheduled again after you have triggered it with the replace option.

More information:

Running the next scheduled Application early (see page 60)

Identifying critical jobs for disaster recovery planning

Objective

In the case of a disaster, certain jobs in an Application must run while other jobs are deemed non-essential and should not be selected to run. As part of your disaster planning, you want to set up the Application in advance so that changes to it are not required in the event of a real, or planned, disaster.

Solution

To identify critical jobs for disaster recovery planning

1. In your calendar, define a special day named DISASTER. As there must be at least one entry in the calendar for this special day to let you simulate properly, reference a date far in the future, for example, September 13, 2024.

   Note: The next DISASTER date should not be more than 25 years in the future to allow for simulations.

2. Use a “Do not run disaster” run frequency condition in the Application for all jobs deemed to be non-essential.

Explanation

Using this solution, you can identify which jobs should not run in the case of a disaster. In the event of a real disaster or a disaster test, you can define another occurrence of the special day, in your calendar, with the appropriate date. When an Application builds, jobs with “Do not run disaster” will not be selected to run.

You can simulate your Applications for the “disaster” date to verify the correct selection of jobs.

Auto-triggering an Event for a reoccurring job

Objective

The Operations department has to run a special standalone job upon request. It is always the same job, and they do not want an operator to have to trigger an Event to run the job.

Solution

To auto-trigger an Event for a reoccurring job

1. Set up a one-job Application. Use a schedule frequency such as Run daily or Run anyday, and define the special job on hold.
2. Use the following JavaScript script at run time for the job:

   ```
   //retrigger same event
   execTrigger('%APPL._event');
   ```

3. Create a Date-Time/Manual Event without a Schedule statement that runs the Application.
4. Manually trigger the Event to create the first generation.

After the first generation of the Application is built, an operator can use the Monitor perspective to release the special job from hold whenever it needs to run. This causes the job to run, the JavaScript script to execute, and the execTrigger built-in function retriggers the same Event to build another generation with the job on hold again.

Explanation

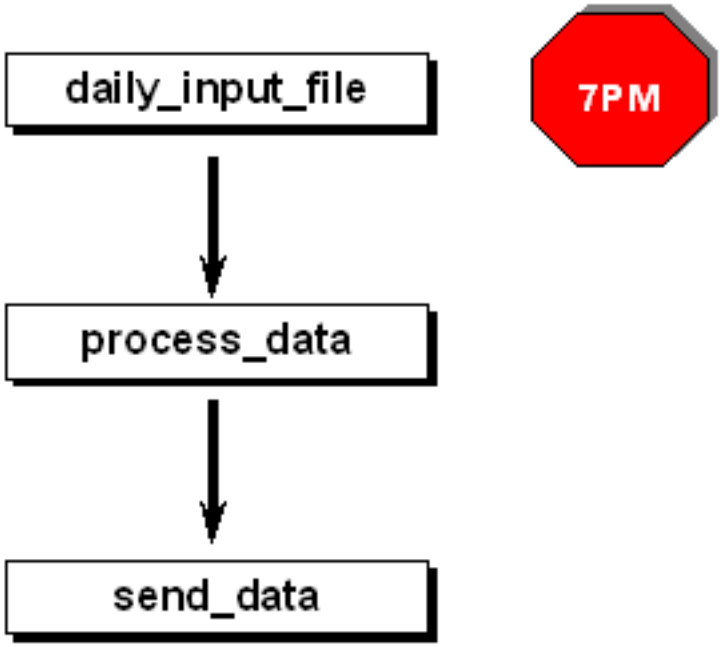
After the Event is triggered and the job is put on hold for the first time, the server automatically triggers another occurrence of the Event every time the operator releases the job.

Completing an Application when a file is not received by its cutoff time

Objective

An Application requires an input file each day before it runs. The cutoff time for this file is 7 p.m. If the input file is not received by this cutoff time, then the Application should be marked complete.

The dependencies look like this:



Solution

To complete an Application when a file is not received by its cutoff time

1. Schedule an Event prior to 7 p.m. to run the Application.
2. Define a file trigger job for the input file, and specify an overdue Not completed by time of “7pm”.
3. Identify an Alert to be triggered if the file trigger becomes overdue.
4. Use the following JavaScript script in the Alert that runs when the file trigger becomes overdue:

```
execCommand('all','%(APPL._name).%APPL._gen','ACTION COMPLETE');
```

Explanation

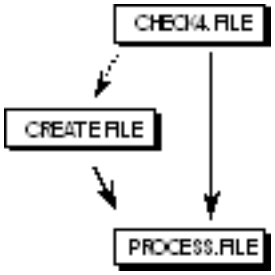
If the file trigger does not complete on time (in other words, by 7 p.m.), the Alert triggers. The Alert runs a JavaScript script to complete the Application. Built-in variables are used for the Application name and generation number to ensure the correct generation is marked complete.

Ensuring a file exists before processing

Objective

Check to ensure a file exists before processing it. If the file does not exist, run a UNIX script to create it.

The dependencies look like this:



Solution

To ensure a file exists before processing

This solution involves one file trigger job and two UNIX scripts. Take the following steps to define the different jobs:

1. Define the file trigger job, CHECK4.FILE, to check for the existence of the file.
   - Specify the name of the file in the File name field. For example:

     `/export/home/jsmith/scripts/myfile`
   - Use the Existing option (Change type) as the trigger condition.
   - On the General dialog, define the job as a conditional job (select the Conditionally option).
2. Define the UNIX job, CREATE.FILE, to create the file if it does not exist.
   - Specify the name of the UNIX script in the Script/Command name field, for example, /export/home/jsmith/scripts/create.file.
   - On the General dialog, define the job as a conditional job (select the Conditionally option).

3. Define the UNIX job, PROCESS.FILE, to process the file.
   - Specify the name of the UNIX script in the Script/Command name field, for example, /export/home/jsmith/scripts/process.file.
   - On the General dialog, specify 1 in the Release job when field.
4. Set up the job relationships and release conditions:
   - Set up CHECK4.FILE to release CREATE.FILE if it fails, and release PROCESS.FILE if it is successful.
   - Set up CREATE.FILE to release PROCESS.FILE if it is successful.

Explanation

This solution uses a file trigger job, CHECK4. FILE, to check for the existence of a file. There are two possible outcomes:

- The file trigger fails. This means the file does not exist. Job CREATE.FILE is released upon failure to create the file. When this job completes, job PROCESS.FILE is released to process the file. At the end of the Application, job CHECK4.FILE is bypassed to let the Application complete.
- The file trigger completes successfully. This means the file exists. Job PROCESS.FILE is released to process the file. At the end of the Application, job CREATE.FILE is bypassed to let the Application complete.

Job PROCESS.FILE sets Release job when to 1 because it can run when either CHECK4.FILE or CREATE.FILE are successful.

CHECK4.FILE and CREATE.FILE are conditional jobs. This lets them be bypassed automatically, if necessary, at the end of the Application.

Using date-qualified file names

Objective

An Application contains some file trigger jobs, where the names of the files to be created contain a date qualifier and a literal in the form *MMDDYY*_ok, where *MM* is the scheduled month, *DD* is the scheduled day, and *YY* is the scheduled year.

Solution

When you define the file trigger jobs, you can use built-in symbolic variables for the date and concatenate the literal at the end. For example, in the File name field you can use the following:

```
/batch/interfaces/file1.%APPL._SMM%APPL._SDD%(APPL._SYY)_ok
```

Note: You need to enclose APPL._SYY in brackets, as the brackets act as delimiters. Otherwise, the server will try to resolve a variable named APPL._SYY_ok.

Explanation

When the Application generates, it resolves the built-in variables based on the scheduled date of the Event. For example, on November 6, 2006, the above example resolves to the following file name:

```
/batch/interfaces/file1.110606_ok
```

Variation

Another approach is to define a symbolic variable, representing the string, in a JavaScript script. You can then use this variable in your job definitions. This is useful when you need to use this file qualifier for many jobs. Also, if the format of the file name changes, you need only to update the format in one place.

For example, use the following JavaScript script at Event trigger time:

```
APPL.datequal = APPL._SMM + APPL._SDD + APPL._SYY + '_ok';
```

When you define a file trigger job, you can use this variable in the File name field, for example:

```
/batch/interfaces/file1.%APPL.datequal
```

Processing a changing file name

Objective

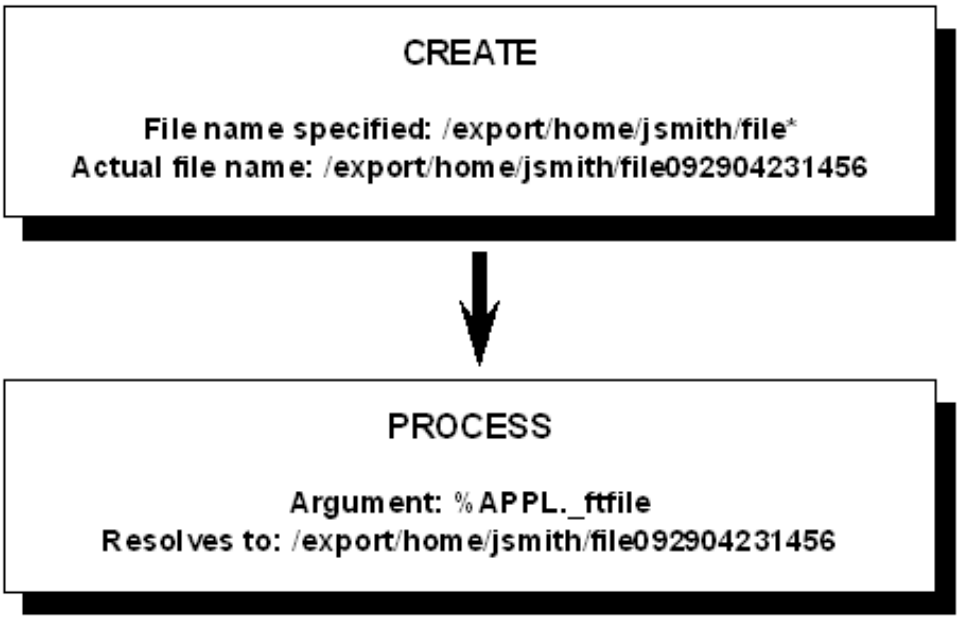
When a file is created, a UNIX script needs to use the file name as input. The last 12 characters of the file name always change and are based on a date and time stamp. An example of such a filename created on 09/29/04 at 23:14:56 is “file092904231456”.

Solution

To process a changing file name

1. Define the file trigger job (for example, CREATE).
   - Specify the name of the file in the File name field. Use an asterisk (*) to mask the file name, for example, /export/home/jsmith/file*.
   - Use the Created option (Change type) as the trigger condition.
2. Define the UNIX script as the successor job (for example, PROCESS). Use %APPL._ftfile in the Arguments to pass field.

   The solution looks like this:



Explanation

The file trigger job waits for a file to be created that matches the file name mask /export/home/jsmith/file*. When any such file is created, the file trigger job completes and releases the successor job to process the file.

The built-in variable, APPL._ftfile, is assigned a value of the full file name whenever file trigger activity takes place. You can pass this variable as an argument to the UNIX script run after the file trigger completes.

There is only one APPL._ftfile variable per generation of an Application. When additional file triggers take place in the same generation of the Application, the previous APPL._ftfile variable is overwritten.

Using the Application name in the script path

Objective

The path to a number of UNIX scripts needs to include the name of the Application in lowercase, for example:

- /export/home/payroll/batch, for the Payroll Application.
- /export/home/billing/batch, for the Billing Application.

Solution

Use the following JavaScript script at run time, either for the Application or for specific jobs within the Application. This script converts the Application name to lowercase using the toLowerCase method of the String class.

```
APPL.applname = APPL._name.toLowerCase();
```

For the location of the script, use %(applname) in the path, for example:

```
/export/home/%(applname)/batch/jobabc
```

Explanation

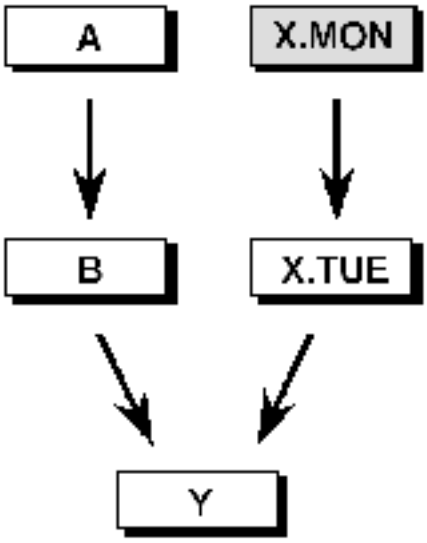
APPL._name is a built-in variable representing the Application name but it resolves to an uppercase name (for example, PAYROLL). This solution uses a JavaScript script to convert the APPL._name variable to a lowercase variable, and you can then use this variable in the path.

Setting up a dependency with a job's previous run

Objective

An Application is scheduled each day. One daily job in the Application, job X, needs to wait for its previous run to complete. However, other jobs in the Application can run even if the previous day's Application is not complete.

The dependencies look like this:



Solution

To set up a dependency with a job's previous run

1. Use a variable as a qualifier for job X that is equal to the first 3 characters of the scheduled day of the week: %APPL._SDAY(1:3).
2. Define the previous run of job X as an “External - Same Scheduler” job.
   a. Use the genTime built-in function to generate date and time variables for the previous day. You can use the following JavaScript script for the job at Event trigger time.

      ```
      APPL.genTime('prev','today less 1 day');
      ```

   b. Use the first 3 characters of the previous day's day-of-week variable (from the genTime above) as a qualifier for the External job:

      ```
      %APPL.prevDAY(1:3).
      ```

   c. Set up this job to release today's run of job X.
   d. Use the External job scheduled: From field on this External job to reflect the previous day (for example, yesterday).
   e. Optionally, use the Home Application name field to specify the name of the Application.

Explanation

Although there is an option to prevent multiple generations of an Application from running at the same time, it cannot be used here. In this scenario, only one job needs to wait.

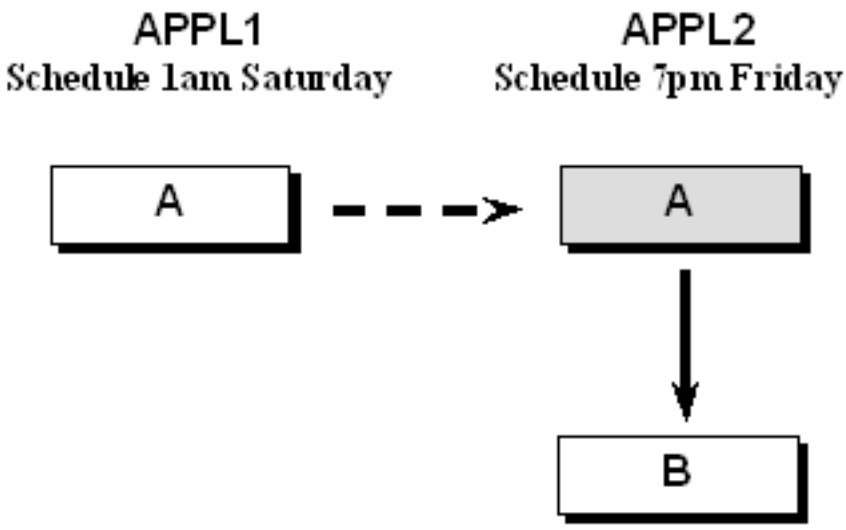
This solution qualifies job X, and sets up the previous day's run as an external dependency to today's run of job X. This ensures job X waits for its previous run to complete, and it does not prevent the other jobs in the Application from running.

Setting up a job with a predecessor scheduled for the next day

Objective

Application APPL1 is scheduled at 1 a.m. on Saturday morning. Application APPL2 is scheduled at 7 p.m. on Friday. Job B in APPL2 runs on Friday and needs to wait for job A in APPL1 that runs the next day.

The dependencies look like this:



Solution

To set up a job with a predecessor scheduled for the next day

After APPL1 has been set up, take the following steps:

1. In APPL2, define job A as an “External - Same Scheduler” job. Set up job A as a predecessor to job B.
2. For the run frequency of job B in APPL2, use Fridays.
3. For the run frequency of job A in APPL2, use Fridays.

   This is the day on which you want to build the job as part of the distant Application.
4. For the job attributes of External job A in APPL2, use a External job scheduled: At time of 1am tomorrow.

   This is the scheduled time for the Event that submits the job in the home Application, APPL1. Optionally, you can specify the name of the Application that submits the job (in other words, APPL1).

Explanation

When APPL2 builds on Friday, it will contain job A as a predecessor to job B. This External job is posted complete when it runs successfully as part of APPL1, which is scheduled for 1 a.m. on Saturday.

Providing notification if a job exceeds its maximum execution time

Objective

Send an email message if a particular job runs longer than 2 hours.

Solution

To provide notification if a job exceeds its maximum execution time

1. Define the job with email notification for the Overdue condition. Customize the email message if you wish.
2. Use the following JavaScript script at run time for the job to reset the job's Late End time.

```
execCommand('%WOB._name','%(APPL._name).%APPL._gen',
 'ACTION RESET LATEEND("now plus 2 hours")');
```

Explanation

When the job is submitted (at run time), the JavaScript script uses the execCommand built-in function to reset the job's end time. The script takes the current time (“now”) and adds the expected execution time of 2 hours.

This solution uses variables for the job name, Application name, and Application generation number to ensure the time for the correct instance of the job is reset.

Bypassing non-critical jobs when jobs are late

Objective

There are problems meeting a service level agreement (SLA) and the programmers recommend bypassing some non-critical jobs when a particular Application is late. For example, if certain jobs in an Application are not complete by 6 a.m., bypass some non-critical jobs.

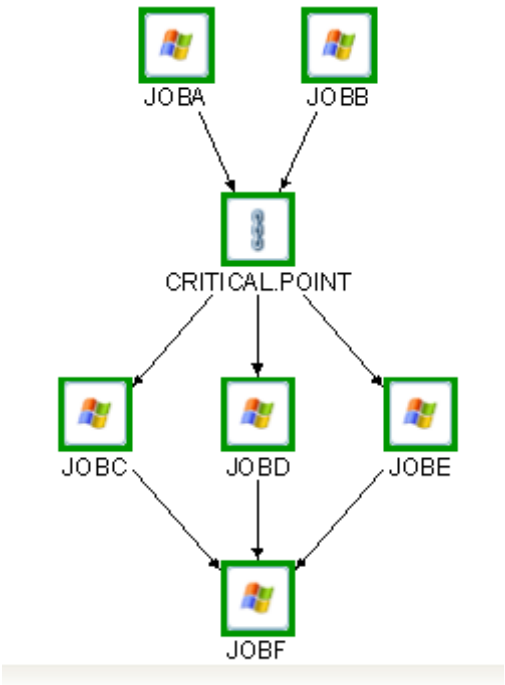
Solution

To bypass non-critical jobs when jobs are late

1. Define a link (for example, CRITICAL.POINT) as a successor to the important jobs.
2. Specify an overdue Not completed by time for this link of 6am.
3. Identify an Alert to be triggered if the link becomes overdue.
4. Define the Alert that runs a JavaScript script to bypass the non-critical jobs, for example:

```
execCommand('JOBC','OPJOBS.%APPL._gen','ACTION BYPASS');
execCommand('JOBE','OPJOBS.%APPL._gen','ACTION BYPASS');
```

The solution looks like this:



Explanation

JOB A and JOB B are important jobs. Each of these jobs releases a link named CRITICAL.POINT. If CRITICAL.POINT is not complete by 6 a.m., it becomes overdue and an Alert triggers. The Alert runs a JavaScript script that uses the execCommand built-in function to bypass jobs C and E.

This solution uses a variable for the Application generation number to ensure the jobs are bypassed in the correct generation of the Application.

Variation

Another solution is to specify a time to bypass job submission for each of the non-critical jobs.

Using the same Application for test and production

Objective

Use the same Application to process work for the test environment and the production environment, even when there are different job requirements for each environment.

Solution

Set up Events to run a common Application. The naming convention for Events is such that the 4-character Event prefix, either PROD or TEST, represents the environment. The first four characters of the job names in the Application are either PROD or TEST.

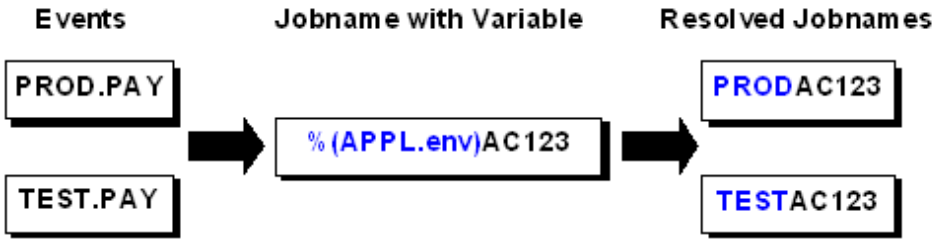
Use the following JavaScript script at Event trigger time to set a variable, APPL.env, based on the Event name, that represents the environment.

```
APPL.env=APPL._event.substring(0,4);
```

You can then use the APPL.env variable in many different ways. Use this variable as part of the Application name, as part of the agent name, and as part of the job names. The following shows some examples:

- Part of the Application name: %(APPL.env)payroll
- Part of the agent name: SUNBOX_%APPL.env
- Part of the job names: %(APPL.env)AC123

The following diagram shows how a job name is resolved based on the Event name:



You can also use “if” logic to handle different requirements for different environments. For example, a job may run in one environment but not in the other. You can use a JavaScript script, like the following script, at Event trigger time to handle conditions like this.

```
if (APPL.env=='PROD') WOB.runthisjob ='daily'
 else WOB.runthisjob = false;
```

Explanation

Different Events for the two environments can run the same Application. The Events use a naming standard that identifies the environment associated with them. A user-defined symbolic variable named APPL.env is defined to represent the environment, and it uses a substring of the built-in variable for the Event name. The APPL.env variable can be used in many places.

If-else logic can be used to determine if a job should run based on the environment for which the schedule is running.

Variation

This example could be extended further by passing the APPL.env variable as an argument to the UNIX scripts being run.

Launching common Applications

Objective

Use the same Application to run a set of jobs for different environments. For example, three sequential jobs (A, B, and C) need to run for different Oracle database instances. Each sequence of jobs requires the name of the agent and the name of the Oracle instance being processed. The name of the Oracle instance is used as follows:

- An argument to the UNIX script being run.
- An environment variable, named ORACLE_SID.

Solution

This solution involves the use of a link in one Application that launches a generic Application for the different environments.

To launch common Applications

1. Set up an Application consisting of one link (for example, LAUNCH.ARCHIVE). This link is used to launch the generic Application, and provides a visible means for tracking that the launch takes place.
2. At Event trigger time, run a JavaScript script for the link with a series of execTrigger built-in functions that trigger the generic Application and pass the variable information as user parameters. For example, the following triggers the PROD.ARCHIVE Event to run the generic Application. It passes the name of the Oracle instance as user parameter 1 and the agent name as user parameter 2:

   ```
   execTrigger('PROD.ARCHIVE','','','','DB1','AGENT01');
   execTrigger('PROD.ARCHIVE','','','','DB99','AGENT02');
   execTrigger('PROD.ARCHIVE','','','','SQL1','TSTAIX32');
   execTrigger('PROD.ARCHIVE','','','','SQL60','TSTLIN54');
   ```
3. Schedule an Event to run this Application.

4. Set up your common Application consisting of jobs A, B, and C. Use the APPL._user*n* variables for the information that changes from one environment to another.

   a. Use the following JavaScript script at Event trigger time and at run time for each job. This script takes the user parameters and assigns them to variables with more specific names.

   ```
   APPL.INSTANCE=APPL._user1;
   APPL.AGENT=APPL._user2;
   ```

   b. Use %AGENT in the Agent name field.

   c. Use %INSTANCE in the Arguments to pass field.

   d. On the Environment Variables dialog, define an environment variable: ORACLE_SID=%INSTANCE.

   Note: You can set these options globally, based on the type of job being defined, to minimize the effort in defining each job.

   e. Optionally, use %AGENT and/or %INSTANCE as part of the Application name or as a job qualifier for each job. This makes it easier to distinguish the jobs for monitoring and control purposes.

5. Define an Event to run your common Application. This Event should not be scheduled since it is triggered from the launch Application.

Explanation

The Application containing a link launches the generic Application several times with the variable data passed as user parameters.

The first execTrigger passes “DB1” as user parameter 1 and “AGENT01” as user parameter 2. The following table shows the fields, variables, and resolved variables based on these user parameters.

Field	Variable	Resolved Variable
Application Name	%(AGENT)_%INSTANCE	AGENT01_DB1
Agent Name	%AGENT	AGENT01
Job Names	A.%INSTANCE	A.DB1
	B.%INSTANCE	B.DB1
	C.%INSTANCE	C.DB1
Arguments to pass	%INSTANCE	DB1
Environment Variable Name and Value	ORACLE_SID=%INSTANCE	ORACLE_SID=DB1

To add another group of jobs, simply add another execTrigger built-in function to the launch Application to launch the generic Application with the new parameters. For example, to use the MYTESTSYS agent to run the same generic Application for the RWP instance of an Oracle database on the agent's computer, add the following:

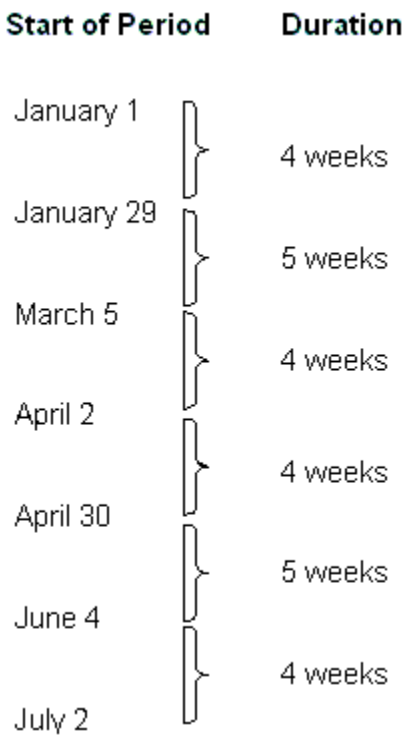
```
execTrigger('PROD.ARCHIVE','','','','RWP','MYTESTSYS');
```

Setting up 4-5-4 periods

Objective

Many jobs are scheduled and parameters are set based on 4-5-4 periods, where there is a 4-week period, followed by a 5-week period, followed by a 4-week period.

The first few periods look like this:



Solution

To set up 4-5-4 periods

From the Special Days section of your calendar definition, take the following steps:

1. Click New.
2. In the Special day name field of the Special day definition dialog, specify a name for the special period (for example, PERIOD454).
3. In the Starting date field, select the starting date of the first 4-5-4 period from the calendar, for example January 1.
4. In the Frequency section, select Repeat.
5. In the Repeat text box, enter the number of 4-5-4 periods to define, for example 4.
6. Enter 13 in the Every text box and select weeks.

   The 4-5-4 sequence repeats every 13 weeks.
7. Click OK.

   You have defined the first 4-week period of each 4-5-4 sequence.
8. Repeat the above steps to define the five-week period of each 4-5-4 sequence. Change the start date to January 29.
9. Repeat the above steps to define the second 4-week period of each 4-5-4 sequence. Change the start date to March 5.
10. Click the Save button to save your special period definition.

Explanation

This solution shows a quick technique for defining the 4-5-4 periods. It uses the fact that the 4-5-4 sequence repeats every 13 weeks. You can define multiple repetitive instances of a special day based on your starting point (January 1, 2005 in this example).

Once you define a special period, you can use it in scheduling criteria. You do not need to know the day of the week or the date to which each statement refers. The server calculates this for you.

- Last workday of each 4-5-4 period:

  ```
  LAST WORKDAY OF PERIOD454
  ```
- First workday of the current 4-5-4 period:

  ```
  1ST WORKDAY OF THIS PERIOD454
  ```
- Last workday of the 2nd week of each 4-5-4 period:

  ```
  LAST WORKDAY OF THE 2ND WEEK OF PERIOD454
  ```

Creating a variable for the accounting year

Objective

Create a variable for the accounting year, to be used whenever you need to reference the current accounting year. For example, from October 4, 2006 until October 1, 2007, inclusively, the accounting year symbolic variable should have the value 2007.

Solution

Use the following JavaScript script to create the variable:

```
APPL.genTime('AC','last day of acct_year');
```

Explanation

In this example, there are a number of special days named ACCT_YEAR. The accounting year is the period between two ACCT_YEAR special days. Each accounting year starts near the beginning of October. For example, the accounting year for the year 2007 begins on October 4, 2006. The accounting year for the year 2008 begins on October 2, 2007.

At any point in the year, you need the server to determine which accounting year you are in. For example, from October 4, 2006 until October 1, 2007 (inclusively), you want an accounting year variable to have the value 2007.

The simplest way to determine this is to use the genTime built-in function to generate date and time variables for the last day of the accounting year. This will always give you the actual accounting year no matter what the date is in the actual year. You can then use the %ACYEAR variable, from the genTime, as you choose.

Calculating the week number

Objective

Many jobs require the week number. The week numbers start from 1 in one calendar year and carry over into the next year. For example, this may be based on a fiscal year that starts 9 days after the 1st Sunday of the year.

Solution

You can use the following JavaScript script to calculate the week number:

```
genTime('ly','today less 1 year');
 genTime('fy','1st sunday of this year plus 9 days');
 n=daysTo(fyDATE);
 if (n > 0) styear=lyYEAR;
 else styear=fyYEAR;
 x='1st sunday of year plus 9 days starting jan 1,' +styear;
 genTime('w',x);
 APPL.weekno=daysBetween(wDATE,'today plus 1 day','tuesday');
```

Explanation

This JavaScript script determines on which year to base the calculation. If the number of days to the first Sunday of the current year plus 9 days is positive, then the current fiscal year started in the previous calendar year. Otherwise, the current fiscal year started this year.

The genTime built-in function generates date and time variables for the 1st day of the current fiscal year. The daysBetween function is then used to calculate the week number, and this number is assigned to a variable named APPL.weekno.

Using date variables for multiple jobs

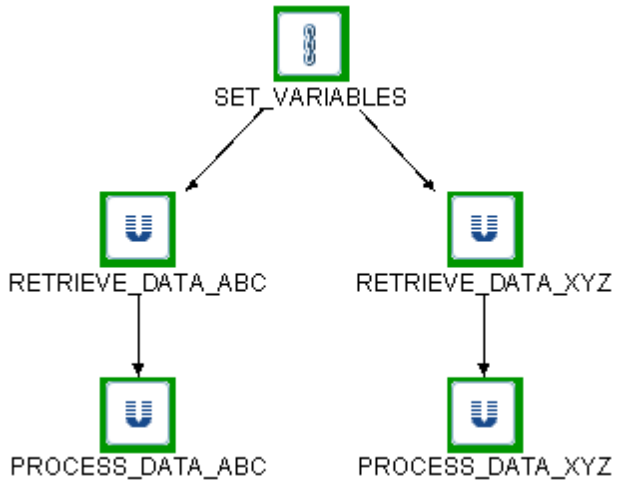
Objective

Many jobs within an Application use the same types of date variables when they run.

Solution

Use a link at the beginning of your Application to calculate the variables needed for the jobs in the Application. Prefix each of these variables with APPL.

For example, you could have an Application like this:



You can define a link named set_variables at the beginning of your Application that runs a JavaScript script at run time. The JavaScript script can set variables, such as the following:

```
APPL.currdate = APPL._SMM + APPL._SDD + APPL._SYY;
APPL.genTime('fdom','first day of this month');
APPL.genTime('nwd','today plus 1 workday');
```

You can then pass these variables as arguments to those jobs that use them. For example, you can pass the following argument in the Arguments to pass field:

```
"%fdomMM %fdomDD %fdomYY" %APPL.currdate %nwdDD
```

Explanation

This is a more efficient approach than running the same JavaScript script over and over for many jobs. By prefixing the variables with “APPL” and running the script at run time, the variables are available when each job within the Application runs.

Building a date parameter

Objective

A job requires a date parameter on the last workday of the month. The format of the parameter consists of the letter M, followed by a series of two-digit day of the month numbers. These numbers range from the current day up to 31 regardless of the number of days in the month.

For example, if the last workday of the month is the 29th day of the month, the date parameter looks like this:

```
M293031
```

Solution

The best approach to create this date parameter involves using a loop in a JavaScript script. The following presents two solutions:

Solution 1

Use the following JavaScript script at run time for the job. This solution uses the “for” statement.

```
APPL.dateparm='M';
 genTime('lwd','last workday of month');
 for (var i=lwdDD; i<=31; i++)
 {
  APPL.dateparm+=i;
 }
```

Solution 2

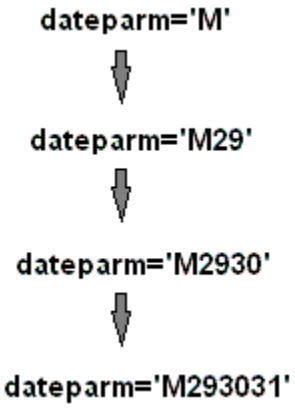
Use the following JavaScript script at run time for the job. This solution uses the “while” statement.

```
APPL.dateparm='M';
 genTime('lwd','last workday of month');
 i=lwdDD;
 while (i<=31)
 {
  APPL.dateparm+=i;
  i++;
 }
```

Explanation

The dateparm variable is initialized to “M”. A loop is used to concatenate each day of the month onto the end of this variable starting with the last workday of the month and ending with “31”.

For example, if the last workday of the month falls on the 29th, the dateparm variable is built as follows:



You can then pass the variable as an argument to the jobs that require it.

Variation

A variation of the objective for this example is to create a similar date parameter, but instead the numbers range from the last workday of the month up to and including the last day of the month. For example, if the last workday of the month is the 27th and the month has only 30 days, then the date parameter is: M27282930.

Use the following JavaScript script at run time for the job. This solution uses the “while” statement.

```
APPL.dateparm='M';
 genTime('lwd','last workday of month');
 genTime('ld','last day of month');
 i=lwdDD;
 while (i<=ldDD)
 {
  APPL.dateparm+=i;
  i++;
 }
```

Creating a variable for the day of week number

Objective

Jobs in an Application use a one-digit day-of-week number that begins with “1” on Wednesday and ends with “7” on Tuesday.

Solution

Use the daysFrom built-in function to define a variable that calculates the number of days from last Tuesday until today. For example, you can use the following JavaScript script at run time:

```
APPL.daynum=daysFrom('tuesday less 1 week');
```

You can then pass this variable as an argument to those jobs that use it.

Explanation

Although CA Workload Automation DE has day-of-week number variables (for example, APPL._SDOW#), they represent the number for the day of the week as follows: 1 for Sunday, 2 for Monday, and so on. This is independent of calendar settings.

This solution defines a variable, APPL.daynum, that represents the number of days since the previous Tuesday. This number corresponds to the day of the week when the week starts on Wednesday, for example:

Day	APPL.daynum
Wednesday	1
Thursday	2
Friday	3
Saturday	4
Sunday	5
Monday	6
Tuesday	7

Incrementing a cycle number

Objective

When an Application processes, it uses a four-digit cycle number. This cycle number is used as part of the file names for two file trigger jobs, and extensively in UNIX scripts that run after the file triggers complete. The cycle number increments each time the Application runs, and it ranges from “0001” to “9999”.

The Application waits for two files to be created: BUS_BC*number* and RES_BC*number*, where *number* represents the cycle number.

Solution

To increment a cycle number

1. The cycle number is passed to the Event using a user parameter. Define the two file trigger jobs. Use the following JavaScript script at Event trigger time for the file triggers to assign the value of the user parameter to a variable named APPL.cycle.

   ```
   APPL.cycle = APPL._user1;
   ```

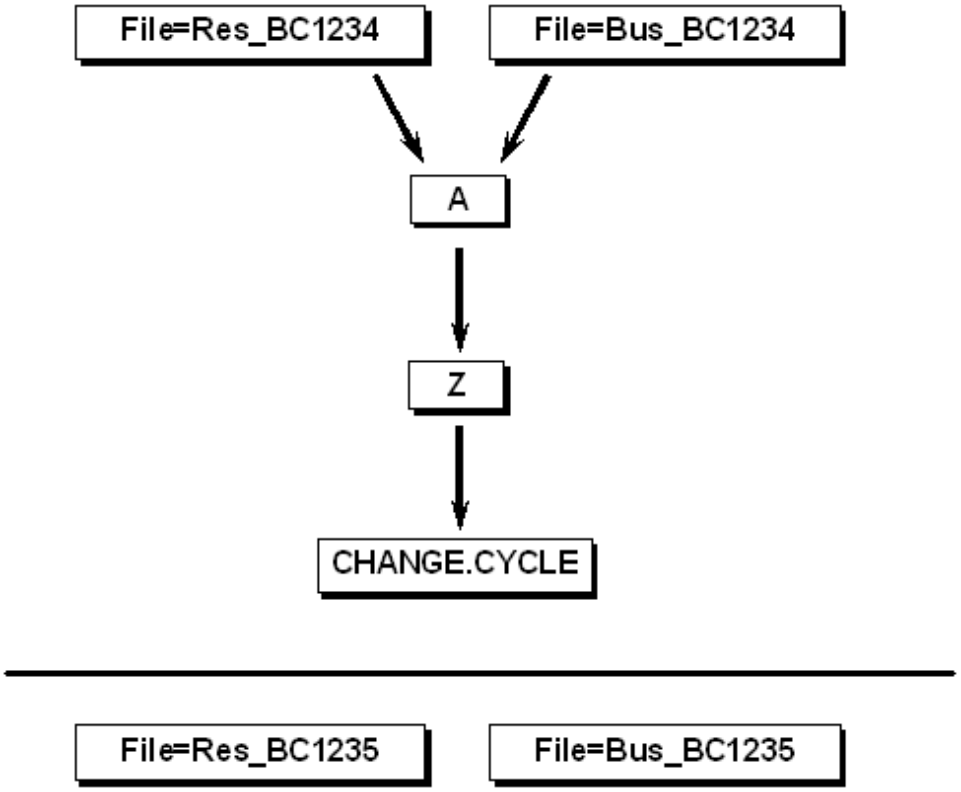
2. Use %cycle as part of the file names for these file triggers, for example, BUS_BC%cycle and RES_BC%cycle.

3. Pass %cycle as an argument to the UNIX scripts that use the cycle number.

4. Use a link at the end of the Application to increment the cycle number and trigger the next occurrence of the Event with the USER1 parameter as the new cycle number. Use the following JavaScript script at run time for the link to achieve this.

   ```
   APPL.cycle = parseInt(APPL.cycle,10);
   if (APPL.cycle == '9999') APPL.next_cycle ='0001';
    else APPL.next_cycle =APPL.cycle + 1;
   APPL.next_cycle=APPL.next_cycle.toString();
   while (APPL.next_cycle.length < 4)
    {
     APPL.next_cycle = '0' + APPL.next_cycle;
    }
   execTrigger('%APPL._event','','','','%APPL.next_cycle');
   ```

The flow looks like this:



You will need to trigger this Event manually the first time and pass the four-digit cycle number. After that, each new generation of the Application will be automatically triggered when one generation completes.

Explanation

The link at the end of the Application takes the current cycle number, adds 1, and pads the result with leading zeroes, as necessary. After cycle “9999”, the cycle is reset to “0001”. The link retriggers the Event and passes this new cycle number in the USER1 field. This lets the new cycle number be used in the next generation of the Application.

Index

D

E

F

G

Q

R

S

T

U

V

W