

DBMS_SCHEDULER

The `DBMS_SCHEDULER` package provides a collection of scheduling functions and procedures that can be called from any PL/SQL program.

This chapter contains the following topics:

- [Deprecated Subprograms](#)
- [Security Model](#)
- [Rules and Limits](#)
- [Operational Notes](#)
- [Data Structures](#)
- [Summary of DBMS_SCHEDULER Subprograms](#)



See Also:

Oracle Database Administrator's Guide for more information regarding how to use `DBMS_SCHEDULER`

DBMS_SCHEDULER Deprecated Subprograms

Oracle recommends that you do not use deprecated subprograms in new applications. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated with Oracle Database 12c Release 1 (12.1):

- [CREATE_CREDENTIAL Procedure](#)
- [DROP_CREDENTIAL Procedure](#)

DBMS_SCHEDULER Security Model

The `DBMS_SCHEDULER` package ignores privileges granted on scheduler objects, such as jobs or chains, through roles. Object privileges must be granted directly to the user.

DBMS_SCHEDULER Rules and Limits

These rules apply when using the `DBMS_SCHEDULER` package.

- Only `SYS` can perform actions on objects in the `SYS` schema.
- Several of the procedures accept comma-delimited lists of object names. If you provide a list of names, then the Scheduler stops executing the list at the first object that returns an error. Therefore, the Scheduler does not perform the tasks needed for the remaining objects on the list.

For example, consider the statement `DBMS_SCHEDULER.STOP_JOB ('job1, job2, job3, sys.jobclass1, sys.jobclass2, sys.jobclass3');`

If `job3` cannot be stopped, then the jobs that follow it, `jobclass1`, `jobclass2`, and `jobclass3` cannot be stopped. The jobs that preceded `job3`, `job1` and `job2`, are stopped.

- Performing an action on an object that does not exist returns a PL/SQL exception stating that the object does not exist.

DBMS_SCHEDULER Operational Notes

The Scheduler uses a rich **calendar**ing syntax to enable you to define repeating schedules, such as "every Tuesday and Friday at 4:00 p.m." or "the second Wednesday of every month." This calendaring syntax is used in calendaring expressions in the `repeat_interval` argument of a number of package subprograms. Evaluating a calendaring expression results in a set of discrete timestamps.

See *Oracle Database Administrator's Guide* for examples of the calendaring syntax.

Calendaring Syntax

This section starts with the calendaring syntax. It is followed by descriptions of various parts of the syntax.

In the calendaring syntax, `*` means 0 or more.

```
repeat_interval = regular_schedule | combined_schedule
```

```
regular_schedule = frequency_clause
[";" interval_clause] [";" bymonth_clause] [";" byweekno_clause]
[";" byyearday_clause] [";" bydate_clause] [";" bymonthday_clause]
[";" byday_clause] [";" byhour_clause] [";" byminute_clause]
[";" bysecond_clause] [";" bysetpos_clause] [";" include_clause]
[";" exclude_clause] [";" intersect_clause] [";" periods_clause]
[";" byperiod_clause]
```

```
frequency_clause = "FREQ" "=" ( predefined_frequency | user_defined_frequency )
predefined_frequency = "YEARLY" | "MONTHLY" | "WEEKLY" | "DAILY" |
    "HOURLY" | "MINUTELY" | "SECONDLY"
user_defined_frequency = named_schedule
```

```
interval_clause = "INTERVAL" "=" intervalnum
    intervalnum = 1 through 99
bymonth_clause = "BYMONTH" "=" monthlist
    monthlist = month ( "," month)*
    month = numeric_month | char_month
    numeric_month = 1 | 2 | 3 ... 12
    char_month = "JAN" | "FEB" | "MAR" | "APR" | "MAY" | "JUN" |
        "JUL" | "AUG" | "SEP" | "OCT" | "NOV" | "DEC"
byweekno_clause = "BYWEEKNO" "=" weeknumber_list
    weeknumber_list = weeknumber ( "," weeknumber)*
    weeknumber = [minus] weekno
    weekno = 1 through 53
byyearday_clause = "BYYEARDAY" "=" yearday_list
    yearday_list = yearday ( "," yearday)*
    yearday = [minus] yeardaynum
    yeardaynum = 1 through 366
bydate_clause = "BYDATE" "=" date_list
    date_list = date ( "," date)*
    date = [YYYY]MMDD [ offset | span ]
```

```

bymonthday_clause = "BYMONTHDAY" "=" monthday_list
    monthday_list = monthday ( "," monthday)*
    monthday = [minus] monthdaynum
    monthdaynum = 1 through 31
byday_clause = "BYDAY" "=" byday_list
    byday_list = byday ( "," byday)*
    byday = [weekdaynum] day
    weekdaynum = [minus] daynum
    daynum = 1 through 53 /* if frequency is yearly */
    daynum = 1 through 5 /* if frequency is monthly */
    day = "MON" | "TUE" | "WED" | "THU" | "FRI" | "SAT" | "SUN"
BYTIME clause: BYTIME=[hour_minute_second_list|minute_second_list]
    hour_minute_second_list: hh24mmss, .., hh24mmss
    minute_second_list: mmss, .. mmss
byhour_clause = "BYHOUR" "=" hour_list
    hour_list = hour ( "," hour)*
    hour = 0 through 23
byminute_clause = "BYMINUTE" "=" minute_list
    minute_list = minute ( "," minute)*
    minute = 0 through 59
bysecond_clause = "BYSECOND" "=" second_list
    second_list = second ( "," second)*
    second = 0 through 59
bysetpos_clause = "BYSETPOS" "=" setpos_list
    setpos_list = setpos ( "," setpos)*
    setpos = [minus] setpos_num
    setpos_num = 1 through 9999

include_clause = "INCLUDE" "=" schedule_list
exclude_clause = "EXCLUDE" "=" schedule_list
intersect_clause = "INTERSECT" "=" schedule_list
schedule_list = schedule_clause ( "," schedule_clause)*
schedule_clause = named_schedule [ offset ]
named_schedule = [schema "."] schedule
periods_clause = "PERIODS" "=" periodnum
byperiod_clause = "BYPERIOD" "=" period_list
period_list = periodnum ( "," periodnum)*
periodnum = 1 through 100

offset = ("+" | "-") ["OFFSET:"] duration_val
span = ("+" | "-" | "^") "SPAN:" duration_val
duration_val = dur-weeks | dur_days
dur_weeks = numofweeks "W"
dur_days = numofdays "D"
numofweeks = 1 through 53
numofdays = 1 through 376
minus = "-"

combined_schedule = schedule_list

```

Table 174-1 Values for repeat_interval

Name	Description
FREQ	This specifies the type of recurrence. It must be specified. The possible predefined frequency values are YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY, and SECONDLY. Alternatively, specifies an existing schedule to use as a user-defined frequency.
INTERVAL	This specifies a positive integer representing how often the recurrence repeats. The default is 1, which means every second for secondly, every day for daily, and so on. The maximum value is 99.

Table 174-1 (Cont.) Values for repeat_interval

Name	Description
BYMONTH	This specifies which month or months you want the job to execute in. You can use numbers such as 1 for January and 3 for March, as well as three-letter abbreviations such as FEB for February and JUL for July.
BYWEEKNO	<p>This specifies the week of the year as a number. It follows ISO-8601, which defines the week as starting with Monday and ending with Sunday; and the first week of a year as the first week, which is mostly within the Gregorian year. The first week is equivalent to the following two variants: the week that contains the first Thursday of the Gregorian year; and the week containing January 4th.</p> <p>The ISO-8601 week numbers are integers from 1 to 52 or 53; parts of week 1 may be in the previous calendar year; parts of week 52 may be in the following calendar year; and if a year has a week 53, parts of it must be in the following calendar year.</p> <p>As an example, in the year 1998, the ISO week 1 began on Monday December 29th, 1997; and the last ISO week (week 53) ended on Sunday January 3rd, 1999. So December 29th, 1997, is in the ISO week 1998-01, and January 1st, 1999, is in the ISO week 1998-53.</p> <p>byweekno is only valid for YEARLY.</p> <p>Examples of invalid specifications are "FREQ=YEARLY; BYWEEKNO=1; BYMONTH=12" and "FREQ=YEARLY; BYWEEKNO=53; BYMONTH=1".</p>
BYYEARDAY	This specifies the day of the year as a number. Valid values are 1 to 366. An example is 69, which is March 10 (31 for January, 28 for February, and 10 for March). 69 evaluates to March 10 for non-leap years and March 9 in leap years. -2 will always evaluate to December 30th independent of whether it is a leap year.
BYDATE	<p>This specifies a list of dates, where each date is of the form [YYYY]MMDD. A list of consecutive dates can be generated by using the SPAN modifier, and a date can be adjusted with the OFFSET modifier. An example of a simple BYDATE clause follows:</p> <p>BYDATE=0115,0315,0615,0915,1215,20060115</p> <p>The following SPAN example is equivalent to BYDATE=0110,0111,0112,0113,0114, which is a span of 5 days starting at 1/10:</p> <p>BYDATE=0110+SPAN:5D</p> <p>The plus sign in front of the SPAN keyword indicates a span starting at the supplied date. The minus sign indicates a span ending at the supplied date, and the "A" sign indicates a span of <i>n</i> days or weeks centered around the supplied date. If <i>n</i> is an even number, it is adjusted up to the next odd number.</p> <p>Offsets adjust the supplied date by adding or subtracting <i>n</i> days or weeks.</p> <p>BYDATE=0205-OFFSET:2W is equivalent to BYDATE=0205-14D (the OFFSET: keyword is optional), which is also equivalent to BYDATE=0122.</p>
BYMONTHDAY	This specifies the day of the month as a number. Valid values are 1 to 31. An example is 10, which means the 10th day of the selected month. You can use the minus sign (-) to count backward from the last day, so, for example, BYMONTHDAY=-1 means the last day of the month and BYMONTHDAY=-2 means the next to last day of the month.
BYDAY	This specifies the day of the week from Monday to Sunday in the form MON, TUE, and so on. Using numbers, you can specify the 26th Friday of the year, if using a YEARLY frequency, or the 4th THU of the month, using a MONTHLY frequency. Using the minus sign, you can say the second to last Friday of the month. For example, -1 FRI is the last Friday of the month.
BYHOUR	This specifies the hour on which the job is to run. Valid values are 0 to 23. As an example, 10 means 10 a.m.

Table 174-1 (Cont.) Values for repeat_interval

Name	Description
BYMINUTE	This specifies the minute on which the job is to run. Valid values are 0 to 59. As an example, 45 means 45 minutes past the chosen hour.
BYSECOND	This specifies the second on which the job is to run. Valid values are 0 to 59. As an example, 30 means 30 seconds past the chosen minute.
BYSETPOS	<p>This selects one or more items, by position, in the list of timestamps that result after the whole calendaring expression is evaluated. It is useful for requirements such as running a job on the last workday of the month. Rather than attempting to express this with the other <code>BY</code> clauses, you can code the calendaring expression to evaluate to a list of every workday of the month, and then add the <code>BYSETPOS</code> clause to select only the last item of that list. Assuming that workdays are Monday through Friday, the syntax would then be:</p> <pre>FREQ=MONTHLY; BYDAY=MON,TUE,WED,THU,FRI; BYSETPOS=-1</pre> <p>Valid values are 1 through 9999. A negative number selects an item from the end of the list (-1 is the last item, -2 is the next to last item, and so on) and a positive number selects from the front of the list. The <code>BYSETPOS</code> clause is always evaluated last. <code>BYSETPOS</code> is only supported with the <code>MONTHLY</code> and <code>YEARLY</code> frequencies.</p> <p>The <code>BYSETPOS</code> clause is applied to the list of timestamps once per frequency period. For example, when the frequency is defined as <code>MONTHLY</code>, the Scheduler determines all valid timestamps for the month, orders that list, and then applies the <code>BYSETPOS</code> clause. The Scheduler then moves on to the next month and repeats the procedure. Assuming a start date of Jun 10, 2004, the example evaluates to: Jun 30, Jul 30, Aug 31, Sep 30, Oct 29, and so on.</p>
INCLUDE	<p>This includes one or more named schedules in the calendaring expression. That is, the set of timestamps defined by each included named schedule is added to the results of the calendaring expression. If an identical timestamp is contributed by both an included schedule and the calendaring expression, it is included in the resulting set of timestamps only once. The named schedules must have been defined with the <code>CREATE_SCHEDULE</code> procedure.</p> <p>This clause only works on a full day and therefore cannot be used with <code>BYHOUR</code>, <code>BYMIN</code>, and <code>BYSECOND</code>.</p>
EXCLUDE	<p>This excludes one or more named schedules from the calendaring expression. That is, the set of timestamps defined by each excluded named schedule is removed from the results of the calendaring expression. The named schedules must have been defined with the <code>CREATE_SCHEDULE</code> procedure.</p> <p>This clause only works on a full day and therefore cannot be used with <code>BYHOUR</code>, <code>BYMIN</code>, and <code>BYSECOND</code>.</p>

Table 174-1 (Cont.) Values for repeat_interval

Name	Description
INTERSECT	<p>This specifies an intersection between the calendaring expression results and the set of timestamps defined by one or more named schedules. Only the timestamps that appear both in the calendaring expression and in one of the named schedules are included in the resulting set of timestamps.</p> <p>For example, assume that the named schedule <code>last_sat</code> indicates the last Saturday in every month, and that for the year 2005, the only months where the last day of the month is also a Saturday are April and December. Assume also that the named schedule <code>end_qtr</code> indicates the last day of each quarter in 2005:</p> <p><code>3/31/2005, 6/30/2005, 9/30/2005, 12/31/2005</code></p> <p>These calendaring expressions result in the dates that follow:</p> <p><code>3/31/2005, 4/30/2005, 6/30/2005, 9/30/2005, 12/31/2005</code></p> <p><code>FREQ=MONTHLY; BYMONTHDAY=-1; INTERSECT=last_sat,end_qtr</code></p> <p>In this example, the terms <code>FREQ=MONTHLY; BYMONTHDAY=-1</code> indicate the last day of each month.</p> <p>This clause only works on a full day and therefore cannot be used with <code>BYHOUR</code>, <code>BYMIN</code>, and <code>BYSECOND</code>.</p>
PERIODS	<p>This identifies the number of periods that together form one cycle of a user-defined frequency. It is used in the <code>repeat_interval</code> expression of the schedule that defines the user-defined frequency. It is mandatory when the <code>repeat_interval</code> expression in the main schedule contains a <code>BYPERIOD</code> clause. The following example defines the quarters of a fiscal year.</p> <p><code>FREQ=YEARLY; BYDATE=0301,0601,0901,1201; PERIODS=4</code></p>
BYPERIOD	<p>This selects periods from a user-defined frequency. For example, if a main schedule names a user-defined frequency schedule that defines the fiscal quarters shown in the previous example, the clause <code>BYPERIOD=2,4</code> in the main schedule selects the 2nd and 4th fiscal quarters.</p>

Combining Schedules

There are two ways to combine schedules:

- Using a combined schedule expression, which is a list of individual schedules

For example, to create a schedule for all company holidays, you provide a list of individual schedules, where each schedule in the list defines a single holiday. The Scheduler evaluates each individual schedule, and then returns a union of the timestamps returned by each individual schedule.
- Embedding other schedules into the main schedule using `include`, `exclude`, and `intersect` clauses

With this method, the embedded schedules inherit certain attributes from the main schedule.

 - Timestamps generated by the `INCLUDE` clause that fall into periods that are skipped by the main schedule are ignored. This is the case when the main schedule skips periods due to the `INTERVAL` clause, the `BYPERIOD` clause, or the `BYMONTH` clause for `freq=monthly`.

- Days that are added by the `INCLUDE` clause follow the hourly/minutes/seconds execution pattern of the main schedule.
- When the `INCLUDE` clause is present, no date-specific defaults are retrieved from the start date (but time-specific defaults can be). (See "[Start Dates and Repeat Intervals](#)", later in this section.) For example, a `repeat_interval` of `FREQ=MONTHLY; INCLUDE=HOLIDAY` executes only on holidays and not on the month/day defaults retrieved from the start date.

The following is an example:

```
BEGIN
dbms_scheduler.create_schedule('embed_sched', repeat_interval =>
'FREQ=YEARLY;BYDATE=0130,0220,0725');
dbms_scheduler.create_schedule('main_sched', repeat_interval =>
'FREQ=MONTHLY;INTERVAL=2;BYMONTHDAY=15;BYHOUR=9,17;INCLUDE=embed_sched');
END;
/
```

In this example, the dates 1/30, 2/20, and 7/25 are added to the main schedule. However, the Scheduler does not include dates that fall in months that are skipped by the `INTERVAL` clause. If the start date of the main schedule is 1/1/2005, then 2/20 is not added. On the dates that are added, the embedded schedule follows the execution pattern of the main schedule: jobs are executed at 9:00 a.m. and 5:00 p.m. on 1/30 and 7/25. If the embedded schedule does not itself have a start date, it inherits the start date from the main schedule.

User-Defined Frequencies

Instead of using predefined frequencies like `DAILY`, `WEEKLY`, `MONTHLY`, and so on, you can create your own frequencies by creating a schedule that returns the start date of each period. For example, the following `repeat_interval` expression is used in a schedule named `fiscal_year` that defines the start of each quarter in a fiscal year:

```
FREQ=YEARLY;BYDATE=0301,0601,0901,1201;PERIODS=4
```

To return the last Wednesday of every quarter, you create a schedule (the "main schedule") that uses the `fiscal_year` schedule as a user-defined frequency:

```
FREQ=fiscal_year;BYDAY=-1WED
```

Periods in a user-defined frequency do not have to be equal in length. In the main schedule, the `BYSETPOS` clause and numbered weekdays are recalculated based on the size of each period. To select dates in specific periods, you must use the `BYPERIOD` clause in the main schedule. To enable this, the schedule that is used as the user-defined frequency must include a `PERIODS` clause, and it must set its start date appropriately. The first date returned by this schedule is used as the starting point of period 1.

As another example, assuming work days are Monday through Friday, to get the last work day of the 2nd and 4th quarters of the fiscal year, the `repeat_interval` clause in the main schedule is the following:

```
FREQ=fiscal_year;BYDAY=MON,TUE,WED,THU,FRI;BYPERIOD=2,4;BYSETPOS=-1
```

Start Dates and Repeat Intervals

The Scheduler retrieves the date and time from the job or schedule start date and incorporates them as defaults into the `repeat_interval`. For example, if the specified frequency is yearly and there is no `BYMONTH` or `BYMONTHDAY` clause in the repeat interval, then the month and day that the job runs on are retrieved from the start date. Similarly, if frequency is monthly but there is no `BYMONTHDAY` clause in the repeat interval, then the day of the month that the job runs on is

retrieved from the start date. If present, BYHOUR, BYMINUTE, and BYSECOND defaults are also retrieved from the start date, and used if those clauses are not specified. Note that if the INCLUDE, EXCLUDE, or INTERSECT clauses are present, no date-related defaults are retrieved from the start date, but time-related defaults are. The following are some examples:

```
start_date:      4/15/05 9:00:00
repeat_interval: freq=yearly
```

is expanded internally to:

```
freq=yearly;bymonth=4;bymonthday=15;byhour=9;byminute=0;bysecond=0
```

The preceding schedule executes on 04/15/05 9:00:00, 04/15/06 9:00:00, 04/15/07 9:00:00, and so on.

For the next example, assume that schedule S1 has a repeat_interval of
FREQ=YEARLY;BYDATE=0701.

```
start_date:      01/20/05 9:00:00
repeat_interval: freq=yearly;include=S1
```

is expanded internally to:

```
freq=yearly;byhour=9;byminute=0;bysecond=0;include=S1
```

Because an INCLUDE clause is present, date-related information is not retrieved from the start date. However, time-specific information is, so the preceding schedule executes on 07/01/05 9:00:00, 07/01/06 9:00:00, 07/01/08 9:00:00, and so on.

General Rules

When using a calendaring expression, consider the following rules:

- For a regular schedule (as opposed to a combined schedule), the calendar string must start with the frequency clause. All other clauses are optional and can be put in any order.
- All clauses are separated by a semicolon, and each clause can be present at most once, with the exception of the include, exclude, and intersect clauses.
- Spaces are allowed between syntax elements and the strings are case-insensitive.
- The list of values for a specific BY clause do not need to be ordered.
- When not enough BY clauses are present to determine what the next date is, this information is retrieved from the start date. For example, "FREQ=YEARLY" with a start date of 02/15/2003 becomes "FREQ=YEARLY;BYMONTH=FEB; BYMONTHDAY=15", which means every year on the 15th of February.

"FREQ=YEARLY;BYMONTH=JAN, JUL" with start date 01/21/2003 becomes

"FREQ=YEARLY;BYMONTH=JAN, JUL;BYMONTHDAY=21", which means every year on January 21 and July 21.

- The byweekno clause is only allowed if the frequency is YEARLY. It cannot be used with other frequencies. When it is present, it will return all days in that week number. If you want to limit it to specific days within the week, you have to add a BYDAY clause. For example, "FREQ=YEARLY;BYWEEKNO=2" with a start date of 01/01/2003 will return:

```
01/06/2003, 01/07/2003, 01/08/2003, 01/09/2003, 01/10/2003, 01/11/2003, 01/12/2003,
01/05/2004, 01/06/2004, 01/07/2004, .... and so on.
```


Note that when the `byweekno` clause is used, it is possible that the dates returned are from a year other than the current year. For example, if returning dates for the year 2004 and the calendar string is `"FREQ=YEARLY;BYWEEKNO=1,53"` for the specified week numbers in 2004, it will return the dates:

```
12/29/03, 12/30/03, 12/31/03, 01/01/04, 01/02/04, 01/03/04, 01/04/04, 12/27/04,
12/28/04, 12/29/04, 12/30/04, 12/31/04, 01/01/05, 01/02/05
```

- For those `BY` clauses that do not have a consistent range of values, you can count backward by putting a "-" in front of the numeric value. For example, specifying `BYMONTHDAY=31` will not give you the last day of every month, because not every month has 31 days. Instead, `BYMONTHDAY=-1` will give you the last day of the month.

This is not supported for `BY` clauses that are fixed in size. In other words, `BYMONTH`, `BYHOUR`, `BYMINUTE`, and `BYSECOND` are not supported.

- The basic values for the `BYDAY` clause are the days of the week. When the frequency is `YEARLY`, or `MONTHLY`, you are allowed to specify a positive or negative number in front of each day of the week. In the case of `YEARLY`, `BYDAY=40MON`, indicates the 40th Monday of the year. In the case of `MONTHLY`, `BYDAY=-2SAT`, indicates the second to last Saturday of the month.

Note that positive or negative numbers in front of the weekdays are not supported for other frequencies and that in the case of yearly, the number ranges from -53 ... -1, 1 ... 53, whereas for the monthly frequency it is limited to -5 ... -1, 1... 5.

If no number is present in front of the weekday it specifies, every occurrence of that weekday in the specified frequency.

- The first day of the week is Monday.
- Repeating jobs with frequencies smaller than daily follow their frequencies exactly across daylight savings adjustments. For example, suppose that a job is scheduled to repeat every 3 hours, the clock is moved forward from 1:00 a.m. to 2:00 a.m., and the last time the job ran was midnight. Its next scheduled time will be 4:00 a.m. Thus, the 3 hour period between subsequent job runs is retained. The same applies when the clock is moved back. This behavior is not the case for repeating jobs that have frequencies of daily or larger. For example, if a repeating job is supposed to be executed on a daily basis at midnight, it will continue to run at midnight if the clock is moved forward or backward. When the execution time of such a daily (or larger frequency) job happens to fall inside a window where the clock is moved forward, the job executes at the end of the window.
- The calendaring syntax does not allow you to specify a time zone. Instead the Scheduler retrieves the time zone from the `start_date` argument. If jobs must follow daylight savings adjustments, then you must specify a region name for the time zone of the `start_date`. For example specifying the `start_date` time zone as `'US/Eastern'` in New York ensures that daylight saving adjustments are automatically applied. If instead, the time zone of the `start_date` is set to an absolute offset, such as `'-5:00'`, then daylight savings adjustments are not followed and your job execution is off by an hour for half the year.
- When `start_date` is `NULL`, the Scheduler determines the time zone for the repeat interval as follows:
 1. It checks whether or not the session time zone is a region name. The session time zone can be set by either:
 - Issuing an `ALTER SESSION` statement, for example:


```
SQL> ALTER SESSION SET time_zone = 'Asia/Shanghai';
```
 - Setting the `ORA_SDTZ` environment variable.

2. If the session time zone is an absolute offset instead of a region name, the Scheduler uses the value of the `DEFAULT_TIMEZONE` Scheduler attribute. For more information, see the [SET_SCHEDULER_ATTRIBUTE Procedure](#).
3. If the `DEFAULT_TIMEZONE` attribute is `NULL`, the Scheduler uses the time zone of `systimestamp` when the job or window is enabled.

BYSETPOS Clause Rules

The following are rules for the `BYSETPOS` clause.

- The `BYSETPOS` clause is the last clause to be evaluated. It is processed after all other `BY` clauses and the `INCLUDE`, `EXCLUDE` and `INTERSECT` clauses have been evaluated.
- The `INTERVAL` clause does not change the size of the period to which the `BYSETPOS` clause is applied. For example, when the frequency is set to monthly and interval is set to 3, the list of timestamps to which `BYSETPOS` is applied is generated from a month, not a quarter. The only impact of the `INTERVAL` clause is to cause months to be skipped. However, you can still select the second to last workday of the quarter like this:

```
FREQ=MONTHLY;INTERVAL=3;BYDAY=MON,TUE,WED,THU,FRI;BYSETPOS=-2
```

provided that you set the start date in the right month. This example returns the next to last workday of a month, and repeats once a quarter.

- To get consistent results, the set to which `BYSETPOS` is applied is determined from the beginning of the frequency period independently of when the evaluation occurs. Whether the Scheduler evaluates

```
FREQ=MONTHLY;BYDAY=MON,TUE,FRI;BYSETPOS=1,3
```

on 01/01/2004 or 01/15/2004, in both cases the expression evaluates to Friday 01/02/2004, and Tuesday 01/06/2004. The only difference is that when the expression is evaluated on 01/15/2004, the Scheduler determines that there are no matches in January because the timestamps found are in the past, and it moves on to the matches in the next month, February.

BYDATE Clause Rules

The following are rules for the `BYDATE` clause.

- If dates in the `BYDATE` clause do not have their optional year component, the job runs on those dates every year.
- The job execution times on the included dates are derived from the `BY` clauses in the calendaring expression. For example, if `repeat_interval` is defined as

```
freq=daily;byhour=8,13,18;byminute=0;bysecond=0;bydate=0502,0922
```

then the execution times on 05/02 and 09/22 are 8:00 a.m., 1:00 p.m., and 6:00 p.m.

EXCLUDE Clause Rules

Excluded dates without a time component are 24 hour periods. All timestamps that fall on an excluded date are removed. In the following example, `jan_fifteen` is a named schedule that resolves to the single date of 01/15:

```
freq=monthly;bymonthday=15,30;byhour=8,13,18;byminute=0;bysecond=0;
exclude=jan_fifteenth
```

In this case, all three instances of the job are removed for 01/15.

OFFSET Rules

You can adjust the dates of individual named schedules by adding positive offsets to them. For example, to execute `JOB2` exactly 15 days after every occurrence of `JOB1`, add `+OFFSET:15D` to the schedule of `JOB1`, as follows:

```
BEGIN
dbms_scheduler.create_schedule('job2_schedule', repeat_interval =>
  'job1_schedule+OFFSET:15D');
END;
/
```

Note that negative offsets to named schedules are not supported.

Example 174-1 Putting It All Together

This example demonstrates the use of user-defined frequencies, spans, offsets, and the `BYSETPOS` and `INCLUDE` clauses. (Note that the `OFFSET:` keyword is optional in an offset clause.)

Many companies in the retail industry share the same fiscal year. The fiscal year starts on the Sunday closest to February 1st, and subsequent quarters start exactly 13 weeks later. The fiscal year schedule for the retail industry can be defined as the following:

```
begin
dbms_scheduler.create_schedule('year_start', repeat_interval=>
  'FREQ=YEARLY;BYDATE=0201^SPAN:1W;BYDAY=SUN');
dbms_scheduler.create_schedule('retail_fiscal_year',
  to_timestamp_tz('15-JAN-2005 12:00:00','DD-MON-YYYY HH24:MI:SS'),
  'year_start,year_start+13w,year_start+26w,year_start+39w;periods=4');
end;
/
```

The following schedule can be used to execute a job on the 5th day off in the 2nd and the 4th quarters of the retail industry. This assumes that Saturday and Sunday are off days as well as the days in the existing `holiday` schedule.

```
begin
dbms_scheduler.create_schedule('fifth_day_off', repeat_interval=>
  'FREQ=retail_fiscal_year;BYDAY=SAT,SUN;INCLUDE=holiday;
  BYPERIOD=2,4;BYSETPOS=5');
end;
/
```

DBMS_SCHEDULER Data Structures

The `DBMS_SCHEDULER` package defines `OBJECT` types and `TABLE` types.

OBJECT Types

- [JOBARG Object Type](#)
- [JOB_DEFINITION Object Type](#)
- [JOBATTR Object Type](#)
- [SCHEDULER\\$_STEP_TYPE Object Type](#)
- [SCHEDULER\\$_EVENT_INFO Object Type](#)
- [SCHEDULER_FILEWATCHER_RESULT Object Type](#)
- [SCHEDULER_FILEWATCHER_REQUEST Object Type](#)

TABLE Types

- [JOBARG_ARRAY Table Type](#)
- [JOB_DEFINITION_ARRAY Table Type](#)
- [JOBATTR_ARRAY Table Type](#)
- [SCHEDULER\\$_STEP_TYPE_LIST Table Type](#)

DBMS_SCHEDULER JOBARG Object Type

This type is used by the `JOB` and `JOBATTR` object types. It represents a job argument in a batch of job arguments.

Syntax

```
TYPE jobarg IS OBJECT (  
    arg_position          NUMBER,  
    arg_text_value       VARCHAR2(4000),  
    arg_anydata_value    ANYDATA,  
    arg_operation        VARCHAR2(5));
```

Attributes

Table 174-2 JOBARG Object Type Attributes

Attribute	Description
<code>arg_position</code>	Position of the argument
<code>arg_text_value</code>	Value of the argument if the type is <code>VARCHAR2</code>
<code>arg_anydata_value</code>	Value of the argument if the type is <code>AnyData</code>
<code>arg_operation</code>	Type of the operation: <ul style="list-style-type: none">• <code>SET</code>• <code>RESET</code>

JOBARG Constructor Function

This constructor function constructs a job argument. It is overloaded to construct job arguments with different types of values.

Syntax

Constructs a job argument with a text value.

```
constructor function jobarg (  
    arg_position          IN POSITIVEN,  
    arg_value             IN VARCHAR2)  
    RETURN SELF AS RESULT;
```

Constructs a job argument with an `AnyData` value.

```
constructor function jobarg (  
    arg_position          IN POSITIVEN,  
    arg_value             IN ANYDATA)  
    RETURN SELF AS RESULT;
```

Constructs a job argument with a `NULL` value.

```
constructor function jobarg (  
    arg_position      IN POSITIVEN,  
    arg_reset         IN BOOLEAN DEFAULT FALSE)  
    RETURN SELF AS RESULT;
```

Parameters

Table 174-3 JOBARG Constructor Function Parameters

Parameter	Description
arg_position	Position of the argument
arg_value	Value of the argument
arg_reset	If arg_reset is TRUE, then the argument at that position is reset. Setting arg_reset to FALSE (which is the default) will create an argument with a NULL value.

JOBARG_ARRAY Table Type

Syntax

```
TYPE jobarg_array IS TABLE OF jobarg;
```

JOBARG_ARRAY Table Type

The `jobarg_array` type is a table of `jobarg`.

Syntax

```
TYPE jobarg_array IS TABLE OF jobarg;
```

DBMS_SCHEDULER JOB_DEFINITION Object Type

This type is used by the `CREATE_JOBS` procedure and represents a job in a batch of jobs.

Syntax

```
TYPE job_definition IS OBJECT (  
    job_name          VARCHAR2(100),  
    job_class         VARCHAR2(32),  
    job_style         VARCHAR2(11),  
    program_name      VARCHAR2(100),  
    job_action        VARCHAR2(4000),  
    job_type          VARCHAR2(20),  
    schedule_name     VARCHAR2(65),  
    repeat_interval   VARCHAR2(4000),  
    schedule_limit    INTERVAL DAY TO SECOND,  
    start_date        TIMESTAMP WITH TIME ZONE,  
    end_date          TIMESTAMP WITH TIME ZONE,  
    event_condition   VARCHAR2(4000),  
    queue_spec        VARCHAR2(100),  
    number_of_arguments NUMBER,  
    arguments         SYS.JOBARG_ARRAY,  
    job_priority      NUMBER,  
    job_weight        NUMBER,  
    max_run_duration  INTERVAL DAY TO SECOND,  
    max_runs          NUMBER,  
    max_failures      NUMBER,
```

```

logging_level          NUMBER,
restartable            VARCHAR2(5),
stop_on_window_close  VARCHAR2(5),
raise_events           NUMBER,
comments              VARCHAR2(240),
auto_drop             VARCHAR2(5),
enabled              VARCHAR2(5),
follow_default_timezone VARCHAR2(5),
parallel_instances    VARCHAR2(5),
aq_job               VARCHAR2(5),
instance_id          NUMBER,
credential_name       VARCHAR2(65),
destination           VARCHAR2(4000),
database_role         VARCHAR2(20),
allow_runs_in_restricted_mode VARCHAR2(5);
restart_on_recovery    BOOLEAN;
restart_on_failure     BOOLEAN;)

```

Object Attributes

[Table 174-4](#) provides brief descriptions of the attributes of the `JOB_DEFINITION` object type. For more complete information about these attributes, see the "[CREATE_JOB Procedure](#)" and the "[SET_ATTRIBUTE Procedure](#)".

Table 174-4 JOB_DEFINITION Object Types

Attribute	Description
job_name	Name of the job
job_class	Name of the job class
job_style	Style of the job: <ul style="list-style-type: none"> REGULAR LIGHTWEIGHT IN_MEMORY_RUNTIME IN_MEMORY_FULL
program_name	Name of the program that the job runs
job_action	Inline action of the job. This is either the code for an anonymous PL/SQL block or the name of a stored procedure, external executable, or chain.
job_type	Job action type ('PLSQL_BLOCK', 'STORED_PROCEDURE', 'EXECUTABLE', 'CHAIN', 'EXTERNAL_SCRIPT', 'SQL_SCRIPT', and 'BACKUP_SCRIPT')
schedule_name	Name of the schedule that specifies when the job has to execute
repeat_interval	Inline time-based schedule
schedule_limit	Maximum delay time between scheduled and actual job start before a job run is canceled
start_date	Start date and time of the job
end_date	End date and time of the job
event_condition	Event condition for event-based jobs
queue_spec	File watcher name or queue specification for event-based jobs
number_of_arguments	Number of job arguments
arguments	Array of job arguments
job priority	Job priority

Table 174-4 (Cont.) JOB_DEFINITION Object Types

Attribute	Description
job_weight	*** Deprecated in Oracle Database 11gR2. Do not change the value of this attribute from the default, which is 1. Weight of the job for parallel execution.
max_run_duration	Maximum run duration of the job
max_runs	Maximum number of runs before the job is marked as completed
max_failures	Maximum number of failures tolerated before the job is marked as broken
logging_level	Job logging level
restartable	Indicates whether the job is restartable (TRUE) or not (FALSE)
stop_on_window_close	Indicates whether the job is stopped when the window that it runs in ends (TRUE) or not (FALSE). Equivalent to the stop_on_window_close job attribute described in the SET_ATTRIBUTE Procedure .
raise_events	State changes that raise events
comments	Comments on the job
auto_drop	If TRUE (the default), indicates that the job should be dropped once completed
enabled	Indicates whether the job should be enabled immediately after creating it (TRUE) or not (FALSE)
follow_default_timezone	If TRUE and if the job start_date is null, then when the default_timezone scheduler attribute is changed, the Scheduler recomputes the next run date and time for this job so that it is in accordance with the new time zone.
parallel_instances	For event-based jobs only. If TRUE, on the arrival of the specified event, the Scheduler creates a new lightweight job to handle that event, so multiple instances of the same event-based job can run in parallel. If FALSE, then an event is discarded if it is raised while the job that handles it is already running,
aq_job	For internal use only
instance_id	The instance ID of the instance that the job must run on For in-memory full jobs, the instance_id value determines in which instance to stop the job; if left NULL, the job is stopped in all instances.
credential_name	The credential to use for a single destination or the default credential for a group of destinations
destination	The name of a single external destination or database destination, or a group name of type external destination or database destination
database_role	In an Oracle Data Guard environment, the database role ('PRIMARY' or 'LOGICAL STANDBY') for which the job runs
allow_runs_in_restricted_mode	If TRUE, the job is permitted to run when the database is in restricted mode, provided that the job owner is permitted to log in during this mode

Table 174-4 (Cont.) JOB_DEFINITION Object Types

Attribute	Description
restart_on_recovery	If set to TRUE for a job and the job is stopped by a database shutdown, then the job is restarted when the database is recovered. If set to FALSE, and the job is stopped by a database shutdown, then the job is marked as stopped when the database is recovered.
restart_on_failure	If set to TRUE for a job and the job fails due to an application error, then the job is retried using the normal Scheduler retry mechanism (after 1 second, after 10 seconds, after 100 seconds, and so on, up to a maximum of 6 times). If all 6 retries fail (after about 30 hours), then the job is marked FAILED. If set to FALSE (the default), a failed job is immediately marked FAILED.

JOB_DEFINITION Constructor Function

This constructor function constructs a `job_definition` object.

Syntax

```

constructor function job_definition (
    job_name          IN      VARCHAR2,
    job_style          IN      VARCHAR2 DEFAULT 'REGULAR',
    program_name       IN      VARCHAR2 DEFAULT NULL,
    job_action         IN      VARCHAR2 DEFAULT NULL,
    job_type           IN      VARCHAR2 DEFAULT NULL,
    schedule_name      IN      VARCHAR2 DEFAULT NULL,
    repeat_interval    IN      VARCHAR2 DEFAULT NULL,
    event_condition    IN      VARCHAR2 DEFAULT NULL,
    queue_spec         IN      VARCHAR2 DEFAULT NULL,
    start_date         IN      TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    end_date           IN      TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    number_of_arguments IN      NATURAL DEFAULT NULL,
    arguments          IN      SYS.JOBARG_ARRAY DEFAULT NULL,
    job_class           IN      VARCHAR2 DEFAULT 'DEFAULT_JOB_CLASS',
    schedule_limit      IN      INTERVAL DAY TO SECOND DEFAULT NULL,
    job_priority        IN      NATURAL DEFAULT NULL,
    job_weight          IN      NATURAL DEFAULT NULL,
    max_run_duration   IN      INTERVAL DAY TO SECOND DEFAULT NULL,
    max_runs            IN      NATURAL DEFAULT NULL,
    max_failures        IN      NATURAL DEFAULT NULL,
    logging_level       IN      NATURALN DEFAULT 64,
    restartable         IN      BOOLEAN DEFAULT FALSE,
    stop_on_window_close IN      BOOLEAN DEFAULT FALSE,
    raise_events        IN      NATURAL DEFAULT NULL,
    comments           IN      VARCHAR2 DEFAULT NULL,
    auto_drop           IN      BOOLEAN DEFAULT TRUE,
    enabled             IN      BOOLEAN DEFAULT FALSE,
    follow_default_timezone IN      BOOLEAN DEFAULT FALSE,
    parallel_instances IN      BOOLEAN DEFAULT FALSE,
    aq_job              IN      BOOLEAN DEFAULT FALSE,
    instance_id         IN      NATURAL DEFAULT NULL,
    credential_name     IN      VARCHAR2 DEFAULT NULL,
    destination        IN      VARCHAR2 DEFAULT NULL,
    database_role       IN      VARCHAR2 DEFAULT NULL,
    allow_runs_in_restricted_mode IN      BOOLEAN DEFAULT FALSE)
RETURN SELF AS RESULT;

```


JOB_DEFINITION_ARRAY Table Type

Syntax

```
TYPE job_definition_array IS TABLE OF job_definition;
```

JOB_DEFINITION_ARRAY Table Type

The type `job_definition_array` is a table of `job_definition`.

Syntax

```
TYPE job_definition_array IS TABLE OF job_definition;
```

JOBATTR Object Type

This type is used by the `SET_JOB_ATTRIBUTES` procedure and represents a job attribute in a batch of job attributes.

Syntax

```
TYPE jobattr IS OBJECT (  
    job_name          VARCHAR2(100),  
    attr_name         VARCHAR2(30),  
    char_value        VARCHAR2(4000),  
    char_value2       VARCHAR2(4000),  
    args_value        JOBARG_ARRAY,  
    num_value         NUMBER,  
    timestamp_value   TIMESTAMP(6) WITH TIME ZONE,  
    interval_value    INTERVAL DAY(2) TO SECOND(6));
```

Attributes

Table 174-5 JOBATTR Object Type Attributes

Attribute	Description
<code>job_name</code>	Name of the job
<code>attr_name</code>	Name of the attribute
<code>char_value</code>	Value of the argument if the type is <code>VARCHAR2</code>
<code>char_value2</code>	Second <code>VARCHAR2</code> attribute value
<code>args_value</code>	Value of the argument if the type is a <code>JOBARG</code> array
<code>num_value</code>	Value of the argument if the type is <code>NUMBER</code>
<code>timestamp_value</code>	Value of the argument if the type is <code>TIMESTAMP WITH TIME ZONE</code>
<code>interval_value</code>	Value of the argument if the type is <code>INTERVAL DAY TO SECOND</code>

JOBATTR Constructor Function

This constructor function constructs a job attribute. It is overloaded to create attribute values of the following types: `VARCHAR2`, `NUMBER`, `TIMESTAMP WITH TIME ZONE`, `INTERVAL DAY TO SECOND`, and an array of `JOBARG` types.

Syntax

```

constructor function jobattr (
    job_name          IN VARCHAR2,
    attr_name         IN VARCHAR2,
    attr_value        IN VARCHAR2,
    attr_value2       IN VARCHAR2 DEFAULT NULL)
RETURN SELF AS RESULT;

constructor function jobattr (
    job_name          IN VARCHAR2,
    attr_name         IN VARCHAR2,
    attr_value        IN [NUMBER, BOOLEAN,
                        TIMESTAMP WITH TIME ZONE,
                        INTERVAL DAY TO SECOND, JOBARG_ARRAY])
RETURN SELF AS RESULT;

constructor function jobattr (
    job_name          IN VARCHAR2,
    attr_name         IN VARCHAR2)
RETURN SELF AS RESULT;

```

Parameters

Table 174-6 JOBATTR Constructor Function Parameters

Parameter	Description
job_name	Name of the job
attr_name	Name of the argument
attr_value	Value of the argument
attr_value2	Most attributes have only one value associated with them, but some can have two. The attr_value2 argument is for this optional second value.

JOBATTR Table Type

Syntax

```
TYPE jobattr_array IS TABLE OF jobattr;
```

JOBATTR_ARRAY Table Type

The type jobattr_array is a table of jobattr.

Syntax

```
TYPE jobattr_array IS TABLE OF jobattr;
```

SCHEDULER\$_STEP_TYPE Object Type

This type is used by RUN_CHAIN to return a list of chain steps with an initial state.

Syntax

```

TYPE scheduler$_step_type IS OBJECT (
    step_name  VARCHAR2(32),
    step_type  VARCHAR2(32));

```

Attributes

Table 174-7 SCHEDULER\$_STEP_TYPE Object Type Attributes

Attribute	Description
step_name	Name of the step
step_type	State of the step

SCHEDULER\$_STEP_TYPE_LIST Table Type

This type is a table of scheduler\$_step_type.

Syntax

```
TYPE scheduler$_step_type_list IS TABLE OF scheduler$_step_type;
```

SCHEDULER\$_EVENT_INFO Object Type

This the datatype of the Scheduler event queue SYS.SCHEDULER\$_EVENT_QUEUE, from which your application consumes job state events raised by the Scheduler.

It is a secure queue owned by SYS.

Syntax

```
TYPE SCHEDULER$_EVENT_INFO IS OBJECT (  
    event_type          VARCHAR2(4000),  
    object_owner        VARCHAR2(4000),  
    object_name         VARCHAR2(4000),  
    event_timestamp     TIMESTAMP WITH TIME ZONE,  
    error_code          NUMBER,  
    error_msg           VARCHAR2(4000),  
    event_status        NUMBER,  
    log_id              NUMBER,  
    run_count           NUMBER,  
    failure_count       NUMBER,  
    retry_count         NUMBER,  
    spare1              NUMBER,  
    spare2              NUMBER,  
    spare3              VARCHAR2(4000),  
    spare4              VARCHAR2(4000),  
    spare5              TIMESTAMP WITH TIME ZONE,  
    spare6              TIMESTAMP WITH TIME ZONE,  
    spare7              RAW(2000),  
    spare8              RAW(2000));
```

Attributes

Table 174-8 SCHEDULER_EVENT_INFO Object Type Attributes

Attribute	Description
event_type	One of "JOB_STARTED", "JOB_SUCCEEDED", "JOB_FAILED", "JOB_BROKEN", "JOB_COMPLETED", "JOB_STOPPED", "JOB_SCH_LIM_REACHED", "JOB_DISABLED", "JOB_CHAIN_STALLED", "JOB_OVER_MAX_DUR". For descriptions of these event types, see Table 174-85 .
object_owner	Owner of the job that raised the event
object_name	Name of the job that raised the event
event_timestamp	Time at which the event occurred
error_code	Applicable only when an error is thrown during job execution. Contains the top-level error code.
error_msg	Applicable only when an error is thrown during job execution. Contains the entire error stack.
event_status	Adds further qualification to the event type. If event_type is "JOB_STARTED," status 1 indicates that it is a normal start, and status 2 indicates that it is a retry. If event_type is "JOB_FAILED," status 4 indicates that it was a failure due to an error that was thrown during job execution, and status 8 indicates that it was an unusual termination of some kind. If event_type is "JOB_STOPPED," status 16 indicates that it was a normal stop, and status 32 indicates that it was a stop with the FORCE option set to TRUE.
log_id	Points to the ID in the scheduler job log from which additional information can be obtained. Note that there need not always be a log entry corresponding to an event. In such cases, log_id is NULL.
run_count	Run count for the job when the event was raised.
failure_count	Failure count for the job when the event was raised.
retry_count	Retry count for the job when the event was raised.
spare1 – spare8	Not currently in use.

SCHEDULER_FILEWATCHER_RESULT Object Type

This is the datatype of a file arrival event message.

You access the event message as a parameter of an event-based job (or a parameter of a program referenced by an event-based job). The message contains information needed to locate and process a file that arrived on a local or remote system.

Syntax

```

TYPE scheduler_filewatcher_result IS OBJECT (
  destination          VARCHAR2(4000),
  directory_path       VARCHAR2(4000),
  actual_file_name     VARCHAR2(4000),
  file_size            NUMBER,
  file_timestamp       TIMESTAMP WITH TIME ZONE,
  ts_ms_from_epoch     NUMBER,
  matching_requests    SYS.SCHEDULER_FILEWATCHER_REQ_LIST);

```

Attributes

Table 174-9 SCHEDULER_FILEWATCHER_RESULT Object Type Attributes

Attribute	Description
destination	Destination at which the file was found, expressed as a host name or IP address.
directory_path	Absolute path of directory in which the file was found.
actual_file_name	Actual name of the file that was found. If the file name specified in the file watcher did not contain wildcards, then this is the same as the name specified in the file watcher.
file_size	Size of the file that was found, in bytes.
file_timestamp	Timestamp assigned to the file when the file watcher considered the file found, based on the minimum file size and steady state duration attributes.
ts_ms_from_epoch	For internal use only.
matching_requests	List of matching requests. This is a TABLE of type objects SCHEDULER_FILEWATCHER_REQUEST. Each matching request corresponds to a file watcher whose destination, directory_path, and file_name attributes matched the arrived file. See "SCHEDULER_FILEWATCHER_REQUEST Object Type" .

SCHEDULER_FILEWATCHER_REQUEST Object Type

This type is returned in the `matching_requests` attribute of the SCHEDULER_FILEWATCHER_RESULT Object Type. Its attributes are similar to the attributes of a file watcher.

Syntax

```

TYPE scheduler_filewatcher_request IS OBJECT (
  owner          VARCHAR2(4000),
  name           VARCHAR2(4000),
  requested_path_name VARCHAR2(4000),
  requested_file_name VARCHAR2(4000),
  credential_owner VARCHAR2(4000),
  credential_name  VARCHAR2(4000),
  min_file_size    NUMBER,
  steady_state_dur NUMBER);

```

Attributes

Table 174-10 SCHEDULER_FILEWATCHER_REQUEST Object Type Attributes

Attribute	Description
owner	Owner of the matched file watcher.
name	Name of the matched file watcher.
requested_path_name	Value of the <code>directory_path</code> attribute of the matched file watcher.
requested_file_name	Value of the <code>file_name</code> attribute of the matched file watcher.
credential_owner	Owner of the credential referenced by the matched file watcher.

Table 174-10 (Cont.) SCHEDULER_FILEWATCHER_REQUEST Object Type Attributes

Attribute	Description
<code>credential_name</code>	Name of the credential referenced by the matched file watcher.
<code>min_file_size</code>	Value of the <code>min_file_size</code> attribute of the matched file watcher.
<code>steady_state_dur</code>	Value of the <code>steady_state_duration</code> attribute of the matched file watcher.

Related Topics

- [SCHEDULER_FILEWATCHER_RESULT Object Type](#)
This is the datatype of a file arrival event message.

Summary of DBMS_SCHEDULER Subprograms

This table lists the `DBMS_SCHEDULER` subprograms and briefly describes them.

Table 174-11 DBMS_SCHEDULER Package Subprograms

Subprogram	Description
ADD_EVENT_QUEUE_SUBSCRIBER Procedure	Adds a user as a subscriber to the Scheduler event queue <code>SYS.SCHEDULER\$_EVENT_QUEUE</code>
ADD_GROUP_MEMBER Procedure	Adds one or more members to an existing group
ADD_JOB_EMAIL_NOTIFICATION Procedure	Adds e-mail notifications for a job for a list of recipients and a list of job state events
ADD_TO_INCOMPATIBILITY Procedure	Adds jobs or programs to an existing incompatibility definition
ALTER_CHAIN Procedure	Alters specified steps of a chain
ALTER_RUNNING_CHAIN Procedure	Alters specified steps of a running chain
CLOSE_WINDOW Procedure	Closes an open window prematurely
COPY_JOB Procedure	Copies an existing job
CREATE_CHAIN Procedure	Creates a chain, which is a named series of programs that are linked together for a combined objective
CREATE_CREDENTIAL Procedure	Creates a credential
CREATE_DATABASE_DESTINATION Procedure	Creates a database destination for use with remote database jobs
CREATE_EVENT_SCHEDULE Procedure	Creates an event schedule, which is a schedule that starts a job based on the detection of an event
CREATE_FILE_WATCHER Procedure	Creates a file watcher, which is a Scheduler object that defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job

Table 174-11 (Cont.) DBMS_SCHEDULER Package Subprograms

Subprogram	Description
CREATE_GROUP Procedure	Creates a group
CREATE_INCOMPATIBILITY Procedure	Creates an incompatibility definition
CREATE_JOB Procedure	Creates a single job
CREATE_JOB_CLASS Procedure	Creates a job class, which provides a way to group jobs for resource allocation and prioritization
CREATE_JOBS Procedure	Creates multiple jobs
CREATE_PROGRAM Procedure	Creates a program
CREATE_RESOURCE Procedure	Specifies resources used by jobs or creates a new resource
CREATE_SCHEDULE Procedure	Creates a schedule
CREATE_WINDOW Procedure	Creates a window, which provides a way to automatically activate different resource plans at different times
DEFINE_ANYDATA_ARGUMENT Procedure	Defines a program argument whose value is of a complex type and must be passed encapsulated in an <code>AnyData</code> object
DEFINE_CHAIN_EVENT_STEP Procedure	Adds or replaces a chain step and associates it with an event schedule or inline event. See also: <code>DEFINE_CHAIN_STEP</code> .
DEFINE_CHAIN_RULE Procedure	Adds a rule to an existing chain
DEFINE_CHAIN_STEP Procedure	Defines a chain step, which can be a program or another (nested) chain. See also: <code>DEFINE_CHAIN_EVENT_STEP</code> .
DEFINE_METADATA_ARGUMENT Procedure	Defines a special metadata argument for the program. You can retrieve specific metadata through this argument.
DEFINE_PROGRAM_ARGUMENT Procedure	Defines a program argument whose value can be passed as a string literal to the program
DISABLE Procedure	Disables a program, job, chain, window, database destination, external destination, file watcher, group, or incompatibility
DROP_AGENT_DESTINATION Procedure	Drops one or more external destinations. Use only when the preferred method of dropping external destinations—unregistering the Scheduler agent with the database—fails.
DROP_CHAIN Procedure	Drops an existing chain
DROP_CHAIN_RULE Procedure	Removes a rule from an existing chain
DROP_CHAIN_STEP Procedure	Drops a chain step
DROP_CREDENTIAL Procedure	Drops a credential

Table 174-11 (Cont.) DBMS_SCHEDULER Package Subprograms

Subprogram	Description
DROP_DATABASE_DESTINATION Procedure	Drops one or more database destinations
DROP_FILE_WATCHER Procedure	Drops one or more file watchers
DROP_GROUP Procedure	Drops one or more groups
DROP_INCOMPATIBILITY Procedure	Drops an existing incompatibility definition
DROP_JOB Procedure	Drops a job or all jobs in a job class
DROP_JOB_CLASS Procedure	Drops a job class
DROP_PROGRAM Procedure	Drops a program
DROP_PROGRAM_ARGUMENT Procedure	Drops a program argument
DROP_SCHEDULE Procedure	Drops a schedule
DROP_WINDOW Procedure	Drops a window
DUMP_IN_MEMORY_TRACE Procedure	Dumps the scheduler in-memory trace buffer of the specified process state object address into the current trace file of the requester process.
ENABLE Procedure	Enables a program, job, chain, window, database destination, external destination, file watcher, or group
END_DETACHED_JOB_RUN Procedure	Ends a running detached job
EVALUATE_CALENDAR_STRING Procedure	Evaluates the calendar string and tells you what the next execution date of a job or window will be
EVALUATE_RUNNING_CHAIN Procedure	Forces reevaluation of the rules of a running chain to trigger any rules for conditions that have been satisfied
GENERATE_JOB_NAME Function	Generates a unique name for a job. This enables you to identify jobs by adding a prefix, so, for example, Sally's jobs would be named <code>sally1</code> , <code>sally2</code> , and so on
GET_AGENT_INFO Function	Returns job information specific to an agent, such as how many are running and so on, depending on the attribute selected
GET_AGENT_VERSION Function	Returns the version string of a Scheduler agent that is registered with the database and is currently running
GET_ATTRIBUTE Procedure	Retrieves the value of an attribute of an object
GET_FILE Procedure	Retrieves a file from a host
GET_SCHEDULER_ATTRIBUTE Procedure	Retrieves the value of a Scheduler attribute
OPEN_WINDOW Procedure	Opens a window prematurely. The window is opened immediately for the duration

Table 174-11 (Cont.) DBMS_SCHEDULER Package Subprograms

Subprogram	Description
PURGE_LOG Procedure	Purges specific rows from the job and window logs
PUT_FILE Procedure	Saves a file to one or more hosts
REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure	Unsubscribes a user from the Scheduler event queue <code>SYS.SCHEDULER\$_EVENT_QUEUE</code>
REMOVE_FROM_INCOMPATIBILITY Procedure	Removes jobs or programs from an incompatibility definition
REMOVE_GROUP_MEMBER Procedure	Removes one or more members from a group
REMOVE_JOB_EMAIL_NOTIFICATION Procedure	Removes e-mail notifications for a job
RESET_JOB_ARGUMENT_VALUE Procedure	Resets the current value assigned to an argument defined with the associated program
RUN_CHAIN Procedure	Immediately runs a chain by creating a run-once job
RUN_JOB Procedure	Runs a job immediately
SET_AGENT_REGISTRATION_PASS Procedure	Sets the agent registration password for a database
SET_ATTRIBUTE Procedure	Changes an attribute of a job, schedule, or other Scheduler object
SET_ATTRIBUTE_NULL Procedure	Changes an attribute of an object to NULL
SET_JOB_ANYDATA_VALUE Procedure	Sets the value of a job argument encapsulated in an <code>AnyData</code> object
SET_JOB_ARGUMENT_VALUE Procedure	Sets the value of a job argument
SET_JOB_ATTRIBUTES Procedure	Sets the value of a job attribute
SET_RESOURCE_CONSTRAINT Procedure	Specifies the resources used by jobs
SET_SCHEDULER_ATTRIBUTE Procedure	Sets the value of a Scheduler attribute
STOP_JOB Procedure	Stops a currently running job or all jobs in a job class

ADD_EVENT_QUEUE_SUBSCRIBER Procedure

This procedure adds a user as a subscriber to the Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE`, and grants the user permission to dequeue from this queue using the designated agent.

Syntax

```
DBMS_SCHEDULER.ADD_EVENT_QUEUE_SUBSCRIBER (
    subscriber_name          IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-12 ADD_EVENT_QUEUE_SUBSCRIBER Procedure Parameters

Parameter	Description
subscriber_name	Name of the Oracle Advanced Queuing (AQ) agent to be used to subscribe to the Scheduler event queue. If NULL, an agent is created and assigned the user name of the calling user.

Usage Notes

The subscription is rule-based. The rule permits the user to see only events raised by jobs that the user owns, and filters out all other messages. If an AQ agent with the same name already exists, an error is raised.

ADD_GROUP_MEMBER Procedure

This procedure adds one or more members to an existing group.

Syntax

```
DBMS_SCHEDULER.ADD_GROUP_MEMBER (  
    group_name      IN VARCHAR2,  
    member          IN VARCHAR2);
```

Parameters

Table 174-13 ADD_GROUP_MEMBER Procedure Parameters

Parameter	Description
group_name	The name of the group.
member	<p>A comma-separated list of members to add to the group. Members must match the group type. A group of the same type can be a member. The Scheduler immediately expands the included group name into its list of members.</p> <p>An error is returned if any of the members do not exist. A member that is already in the group is skipped, and no error is generated.</p> <p>The keyword LOCAL can be included as a member for database destination or external destination groups. See the "CREATE_GROUP Procedure" for information about this keyword.</p>

Usage Notes

The following users may add members to a group:

- The group owner
- A user that has been granted the ALTER object privilege on the group
- A user with the CREATE ANY JOB system privilege

You must have the MANAGE_SCHEDULER privilege to add a member to a group of type WINDOW.



See Also:

"CREATE_GROUP Procedure"

ADD_JOB_EMAIL_NOTIFICATION Procedure

This procedure adds e-mail notifications for a job. E-mails are then sent to the specified list of recipients whenever any of the specified job state events is raised.

Syntax

```
DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
  job_name          IN VARCHAR2,
  recipients        IN VARCHAR2,
  sender            IN VARCHAR2 DEFAULT NULL,
  subject           IN VARCHAR2 DEFAULT DBMS_SCHEDULER.DEFAULT_NOTIFICATION_SUBJECT,
  body             IN VARCHAR2 DEFAULT DBMS_SCHEDULER.DEFAULT_NOTIFICATION_BODY,
  events            IN VARCHAR2 DEFAULT 'JOB_FAILED,JOB_BROKEN,JOB_SCH_LIM_REACHED,
                                     JOB_CHAIN_STALLED,JOB_OVER_MAX_DUR',
  filter_condition  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-14 ADD_JOB_EMAIL_NOTIFICATION Procedure Parameters

Parameter	Description
job_name	Name of the job that e-mail notifications are added for. Cannot be NULL.
recipients	Comma-separated list of e-mail addresses to send notifications to. E-mail notifications for all listed events are sent to all recipients. Cannot be NULL.
sender	e-mail address to use as the sender address (the From: address) in the e-mail header. If NULL or omitted, the e-mail address specified in the Scheduler attribute <code>email_sender</code> is used. See <i>Oracle Database Administrator's Guide</i> for more information on this Scheduler attribute.
subject	The subject to use in the e-mail header. Table 174-15 describes the variables that you can include within this parameter. The Scheduler assigns values to these variables before sending the notification. If <code>subject</code> is omitted, the default subject is used. The default subject is the following text, where text enclosed in the '%' character represents a variable: 'Oracle Scheduler Job Notification - %job_owner%.%job_name%.%job_subname% %event_type%'

Table 174-14 (Cont.) ADD_JOB_EMAIL_NOTIFICATION Procedure Parameters

Parameter	Description
body	<p>The body of the e-mail message. Table 174-15 describes the variables that you can include within this parameter. The Scheduler assigns values to these variables before sending the notification. If <code>body</code> is omitted, the default body is used. The default body is the following text, where text enclosed in the '%' character represents a variable:</p> <pre>'Job: %job_owner%.%job_name%.%job_subname% Event: %event_type% Date: %event_timestamp% Log id: %log_id% Job class: %job_class_name% Run count: %run_count% Failure count: %failure_count% Retry count: %retry_count% Error code: %error_code% Error message: %error_message%'</pre>
events	<p>Comma-separate list of job state events to send e-mail notifications for. Cannot be NULL. A notification is sent to all recipients if any of the listed events is raised. Table 174-85 lists the valid events for this parameter. If <code>events</code> is omitted, notifications are sent for the following default events:</p> <pre>JOB_FAILED, JOB_BROKEN, JOB_SCH_LIM_REACHED, JOB_CHAIN_STALLED, JOB_OV ER_MAX_DUR</pre>
filter_condition	<p>Used to filter events to send e-mail notifications for. If NULL, all occurrences of the specified events cause e-mail notifications to be sent. <code>filter_condition</code> must be a boolean SQL WHERE clause that may refer to the <code>:event</code> bind variable. This bind variable is automatically bound to an object of type <code>SCHEDULER\$_EVENT_INFO</code> that represents the raised event.</p> <p>For example, to send an e-mail notification only when the error number in an event is 600 or 700, use the following <code>filter_condition</code>:</p> <pre>:event.error_code=600 or :event.error_code=700</pre> <p>See "SCHEDULER\$_EVENT_INFO Object Type".</p>

[Table 174-15](#) lists the variables that you can use in the subject and body arguments.

Table 174-15 Variables Used in the SUBJECT and BODY Parameters

Variable	Comment
%job_owner%	Schema in which job was created
%job_name%	Name of the job that e-mail notifications are added for
%job_subname%	Present for event-based jobs with the <code>parallel_instances</code> attribute set and for chain steps
%event_type%	Valid values are listed in Table 174-85
%event_timestamp%	Time at which the event occurred
%log_id%	Refers to the <code>LOG_ID</code> column in views <code>*_SCHEDULER_JOB_LOG</code> and <code>*_SCHEDULER_JOB_RUN_DETAILS</code>
%error_code%	Number of the error code.

Table 174-15 (Cont.) Variables Used in the SUBJECT and BODY Parameters

Variable	Comment
%error_message%	The text of the error message
%run_count%	Run count for the job when the event was raised
%failure_count%	Failure count for the job when the event was raised
%retry_count%	Retry count for the job when the event was raised

Usage Notes

You can call `ADD_JOB_EMAIL_NOTIFICATION` once for each different set of notifications that you want to configure for a particular job. For example, you may want to send notifications for the `JOB_FAILED`, `JOB_BROKEN`, `JOB_SCH_LIM_REACHED`, and `JOB_CHAIN_STALLED` events to the principle DBA and all senior DBAs, but send a notification for the `JOB_OVER_MAX_DUR` event only to the principle DBA.

This procedure succeeds only if the Scheduler attribute `email_server` is set to a valid SMTP server. See *Oracle Database Administrator's Guide* for more information.

To call this procedure, you must be the job owner or have the `CREATE ANY JOB` system privilege or have the `ALTER` object privilege on the job.

ADD_TO_INCOMPATIBILITY Procedure

This procedure adds jobs or programs to an existing incompatibility definition.

Syntax

```
DBMS_SCHEDULER.ADD_TO_INCOMPATIBILITY (  
    incompatibility_name    IN VARCHAR2,  
    object_name             IN VARCHAR2);
```

Parameters

Table 174-16 ADD_TO_INCOMPATIBILITY Procedure Parameters

Parameter	Description
<code>incompatibility_name</code>	The name of the incompatibility definition.
<code>object_name</code>	One or more (comma-separated) programs or jobs

Usage Notes

This procedure does not raise an error if any specified objects already exist in the incompatibility definition.



See Also:

Using Incompatibility Definitions in *Oracle Database Administrator's Guide*

ALTER_CHAIN Procedure

This procedure alters an attribute of the specified steps of a chain. This affects all future runs of the specified steps, both in the currently running chain job and in future runs of the same chain job or other chain jobs that point to the chain.

Syntax

Alters the value of a boolean attribute of one or more steps:

```
DBMS_SCHEDULER.ALTER_CHAIN (
    chain_name      IN VARCHAR2,
    step_name       IN VARCHAR2,
    attribute        IN VARCHAR2,
    value           IN BOOLEAN);
```

Alters the value of a character attribute of one or more steps:

```
DBMS_SCHEDULER.ALTER_CHAIN (
    chain_name      IN VARCHAR2,
    step_name       IN VARCHAR2,
    attribute        IN VARCHAR2,
    char_value      IN VARCHAR2);
```

Parameters

Table 174-17 ALTER_CHAIN Procedure Parameters

Parameter	Description
chain_name	The name of the chain to alter
step_name	The name of the step or a comma-separated list of steps to alter. This cannot be NULL.

Table 174-17 (Cont.) ALTER_CHAIN Procedure Parameters

Parameter	Description
attribute	<p>The attribute of the steps to change. Must be one of the following:</p> <ul style="list-style-type: none"> • 'PAUSE' <p>If set to TRUE for a step, after the step has run, its state changes to PAUSED (and the completed attribute remains FALSE).</p> <p>If PAUSE is reset to FALSE for a paused chain step (using ALTER_RUNNING_CHAIN), the state is set to its completion state (SUCCEEDED, FAILED, or STOPPED) and the completed attribute is set to TRUE.</p> <p>Setting PAUSE has no effect on steps that have already run. This allows execution of a chain to be suspended after the execution of certain steps.</p> • 'PAUSED_BEFORE' <p>If set to TRUE for a step and if any of the rule conditions that start the step are true, then its state changes to PAUSED and the step does not run.</p> <p>If PAUSED_BEFORE is reset to FALSE for a chain step that has paused before starting (using ALTER_RUNNING_CHAIN), then the step starts running if any of the rule conditions that start the step are true.</p> <p>Setting PAUSED_BEFORE has no effect on steps that are running or have already run. This allows execution of a chain to be suspended before the execution of certain steps.</p> • 'SKIP' <p>If set to TRUE for a step, when the step condition is met, instead of being run, the step is treated as if it has immediately succeeded. Setting SKIP to TRUE has no effect for a step that is running, scheduled to run after a delay, or has already run. If SKIP is set TRUE for a step that PAUSE is also set for, when the step condition is met, the step immediately changes to state PAUSED.</p> • 'RESTART_ON_FAILURE' <p>If set to TRUE for a step and the step fails due to an application error, then the step is retried using the normal Scheduler retry mechanism (after 1 second, after 10 seconds, after 100 seconds, and so on, up to a maximum of 6 times). If all 6 retries fail (after about 30 hours), then the chain step is marked FAILED.</p> <p>If set to FALSE (the default), a failed chain step is immediately marked FAILED.</p> • 'RESTART_ON_RECOVERY' <p>If set to TRUE for a step and the step is stopped by a database shutdown, then the step is restarted when the database is recovered.</p> <p>If set to FALSE, and the step is stopped by a database shutdown, then the step is marked as stopped when the database is recovered and the chain continues.</p> • 'DESTINATION_NAME' <p>The name of an existing database destination or external destination. You can view external destination names in the view ALL_SCHEDULER_EXTERNAL_DESTS, and database destination names in the views *_SCHEDULER_DB_DESTS. You cannot specify a destination group for this attribute. This parameter is NULL by default.</p> • 'CREDENTIAL_NAME' <p>The credential to use when running this step. NULL by default.</p>
value	The value to set for the attribute (for a boolean attribute).
char_value	The value to set for the attribute (for a character attribute).

Usage Notes

Altering a chain requires `ALTER` privileges on the chain either by being the owner of the chain, or by having the `ALTER` object privilege on the chain or by having the `CREATE ANY JOB` system privilege.

ALTER_RUNNING_CHAIN Procedure

This procedure alters an attribute of the specified steps of a chain. This affects only steps of the instance of the chain for the specified running chain job.

Syntax

```
DBMS_SCHEDULER.ALTER_RUNNING_CHAIN (  
    job_name          IN VARCHAR2,  
    step_name         IN VARCHAR2,  
    attribute         IN VARCHAR2,  
    value             IN {BOOLEAN|VARCHAR2});
```

Parameters

Table 174-18 ALTER_RUNNING_CHAIN Procedure Parameters

Parameter	Description
job_name	The name of the job that is running the chain
step_name	The name of the step or a comma-separated list of steps to alter. If this is set to <code>NULL</code> and attribute is <code>PAUSE</code> or <code>SKIP</code> , then all steps of the running chain are altered.
attribute	<p>The attribute of the steps to change. Valid values are:</p> <ul style="list-style-type: none">'PAUSE' <p>If the <code>PAUSE</code> attribute is set <code>TRUE</code> for a step, then after the step runs, its state changes to <code>PAUSED</code> (and the <code>completed</code> attribute remains false).</p><p>If <code>PAUSE</code> is reset to <code>FALSE</code> for a paused chain step (using <code>ALTER_RUNNING_CHAIN</code>), the state is set to completion (<code>SUCCEEDED</code>, <code>FAILED</code>, or <code>STOPPED</code>) and the <code>completed</code> attribute is set to <code>TRUE</code>. Setting <code>PAUSE</code> has no effect on steps that have already run. This allows execution of a chain to be suspended after the execution of certain steps. If <code>step_name</code> is set to <code>NULL</code>, <code>PAUSE</code> is set to <code>TRUE</code> for all steps of this running chain.</p>'PAUSE_BEFORE' <p>If set to <code>TRUE</code> for a step that has not yet run and if any of the rule conditions that start the step are true, then its state changes to <code>PAUSED</code> and the step does not run.</p><p>If <code>PAUSE_BEFORE</code> is reset to <code>FALSE</code> for a chain step that has paused before starting, then the step starts running if any of the rule conditions that start the step are true.</p><p>Setting <code>PAUSE_BEFORE</code> has no effect on steps that are running or have already run. This allows execution of a chain to be suspended before the execution of certain steps.</p><p>If <code>step_name</code> is set to <code>NULL</code>, then <code>PAUSE_BEFORE</code> is set to the specified value for all steps of this running chain.</p>

Table 174-18 (Cont.) ALTER_RUNNING_CHAIN Procedure Parameters

Parameter	Description
attribute CONTINUED	<ul style="list-style-type: none"> 'SKIP' If the SKIP attribute is set to TRUE for a step, when the step condition is met, instead of being run, the step is treated as if it has immediately succeeded. Setting SKIP to TRUE has no effect for a step that is running, scheduled to run after a delay, or has already run. If step_name is set to NULL, SKIP is set TRUE for all steps of this running chain. If SKIP is set TRUE for a step that PAUSE is also set for, when the step condition is met the step immediately changes to state PAUSED. 'RESTART_ON_FAILURE' If set to TRUE for a step and the step fails due to an application error, then the step is retried using the normal Scheduler retry mechanism (after 1 second, after 10 seconds, after 100 seconds, and so on, up to a maximum of 6 times). If all 6 retries fail (after about 30 hours), then the chain step is marked FAILED. If set to FALSE (the default), a failed chain step is immediately marked FAILED. 'RESTART_ON_RECOVERY' If the RESTART_ON_RECOVERY attribute is set to TRUE for a step, then if the step is stopped by a database shutdown, it is restarted when the database is recovered. If set to FALSE, then if the step is stopped by a database shutdown, the step is marked as stopped when the database is recovered and the chain continues. 'STATE' This changes the state of the steps. The state can only be changed if the step is not running. The state can only be changed to one of the following: 'NOT_STARTED', 'SUCCEEDED', 'FAILED error_code' If the state is being changed to FAILED, an error code must be included (this must be a positive integer).
value	The value to set for the attribute. Valid values are: TRUE, FALSE, 'NOT_STARTED', 'SUCCEEDED', or 'FAILED error_code'

Usage Notes

Altering a running chain requires you to have alter privileges on the job that is running (either as the owner, or as a user with ALTER privileges on the job or the CREATE ANY JOB system privilege).

When trying to update a step defined with a nested chain, it is necessary to specify the job_name as <SCHEMA>.<JOB_NAME>.<STEP_NAME_IN_TOP_LEVEL_CHAIN> to be able to make reference to the steps inside the subchain.

CLOSE_WINDOW Procedure

This procedure closes an open window prematurely. A closed window means that it is no longer in effect. When a window is closed, the Scheduler switches the resource plan to the one that is in effect outside the window, or in the case of overlapping windows, to another window.

Syntax

```
DBMS_SCHEDULER.CLOSE_WINDOW (  
    window_name          IN VARCHAR2);
```

Parameters

Table 174-19 CLOSE_WINDOW Procedure Parameters

Parameter	Description
window_name	The name of the window

Usage Notes

If you try to close a window that does not exist or is not open, an error is generated.

A job that is running does not stop when the window it is running in closes, unless the attribute `stop_on_window_close` is set to `TRUE` for the job. However, the resources allocated to the job can change if the resource plan changes.

When a running job has a group of type `WINDOW` as its schedule, the job is not stopped when its window is closed if another window in the same window group becomes active. This is the case even if the job has the attribute `stop_on_window_close` set to `TRUE`.

Closing a window requires the `MANAGE SCHEDULER` privilege.

COPY_JOB Procedure

This procedure copies all attributes of an existing job to a new job. The new job is created disabled, while the state of the existing job is unaltered.

Syntax

```
DBMS_SCHEDULER.COPY_JOB (  
    old_job              IN VARCHAR2,  
    new_job              IN VARCHAR2);
```

Parameters

Table 174-20 COPY_JOB Procedure Parameters

Parameter	Description
old_job	The name of the existing job
new_job	The name of the new job

Usage Notes

To copy a job, you must have privileges to create a job in the schema of the new job (the `CREATE JOB` system privilege if it is in your own schema, otherwise, the `CREATE ANY JOB` system privilege). If the old job is not in the your own schema, then you must also have `ALTER` privileges on the old job or the `CREATE ANY JOB` system privilege.

CREATE_CHAIN Procedure

This procedure creates a new chain. The chain name can be optionally qualified with a schema name (for example, `myschema.myname`).

A chain is always created as disabled and must be enabled with the [ENABLE Procedure](#) before it can be used.

Syntax

```
DBMS_SCHEDULER.CREATE_CHAIN (  
    chain_name          IN VARCHAR2,  
    rule_set_name       IN VARCHAR2 DEFAULT NULL,  
    evaluation_interval IN INTERVAL DAY TO SECOND DEFAULT NULL,  
    comments            IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-21 CREATE_CHAIN Procedure Parameters

Parameter	Description
chain_name	The name to assign to the new chain, which can optionally be qualified with a schema. This must be unique in the SQL namespace, therefore, there cannot already be a table or other object with this name and schema.
rule_set_name	<p>In the normal case, no rule set should be passed in. The Scheduler automatically creates a rule set and associated empty evaluation context. You then use <code>DEFINE_CHAIN_RULE</code> to add rules and <code>DROP_CHAIN_RULE</code> to remove them.</p> <p>Advanced users can create a rule set that describes their chain dependencies and pass it in here. This allows greater flexibility in defining rules. For example, conditions can refer to external variables, and tables can be exposed through the evaluation context. If you pass in a rule set, you must ensure that it is in the format of a chain rule set. (For example, all steps must be listed as variables in the evaluation context). If no rule set is passed in, the rule set created is of the form <code>SCHED_RULESET\${N}</code> and the evaluation context created is of the form <code>SCHED_EVCTX\${N}</code></p>
evaluation_interval	If this is <code>NULL</code> , reevaluation of the rules of a running chain are performed only when the job starts and when a step completes. A non- <code>NULL</code> value causes rule evaluations to also occur periodically at the specified interval. Because evaluation may be CPU-intensive, this should be conservatively set to the highest possible value or left at <code>NULL</code> if possible. <code>evaluation_interval</code> cannot be less than a minute or greater than a day.
comments	An optional comment describing the purpose of the chain

Usage Notes

To create a chain in your own schema, you must have the `CREATE JOB` system privilege. To create a chain in a different schema you must have the `CREATE ANY JOB` system privilege. If you do not provide a `rule_set_name`, a rule set and evaluation context is created in the schema

that the chain is being created in, so you must have the privileges required to create these objects. See the `DBMS_RULE_ADM.CREATE_RULE_SET` and `DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT` procedures for more information.

CREATE_CREDENTIAL Procedure

This deprecated procedure creates a stored username/password pair. Credentials are assigned to jobs so that they can authenticate with a local or remote host operating system or a remote Oracle database.



Note:

This procedure is deprecated with Oracle Database 12c Release 1 (12.1). While the procedure remains available in this package, for reasons of backward compatibility, Oracle recommends using the alternative enhanced functionality provided in the [DBMS_CREDENTIAL](#) package, specifically the [CREATE_CREDENTIAL Procedure](#).

Syntax

```
DBMS_SCHEDULER.CREATE_CREDENTIAL (
  credential_name      IN VARCHAR2,
  username             IN VARCHAR2,
  password             IN VARCHAR2,
  database_role        IN VARCHAR2 DEFAULT NULL,
  windows_domain       IN VARCHAR2 DEFAULT NULL,
  comments             IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-22 CREATE_CREDENTIAL Procedure Parameters

Parameter	Description
<code>credential_name</code>	The name to assign to the credential. It can optionally be prefixed with a schema name. It cannot be set to <code>NULL</code> . It is converted to uppercase unless enclosed in double quotation marks.
<code>username</code>	The user name for logging into to the host operating system or remote Oracle database. This cannot be set to <code>NULL</code> and is case-sensitive. It cannot contain double quotes or spaces. Maximum length is 64.
<code>password</code>	The password for the user name. This cannot be set to <code>NULL</code> and is case sensitive. The password is stored obfuscated and is not displayed in the Scheduler dictionary views. Maximum length is 128.
<code>database_role</code>	The value of the <code>database_role</code> attribute is used as the system privilege for logging into a remote database to run a remote database job. Valid values are: <code>SYSDBA</code> and <code>SYSOPER</code>
<code>windows_domain</code>	For a Windows remote executable target, this is the domain that the specified user belongs to. The domain is converted to uppercase automatically. Maximum length is 64.
<code>comments</code>	A text string that can be used to describe the credential. Scheduler does not use this parameter. Maximum length is 240.

Usage Notes

Credentials reside in a particular schema and can be created by any user with the `CREATE JOB` system privilege. To create a credential in a schema other than your own, you must have the `CREATE ANY JOB` privilege.

CREATE_DATABASE_DESTINATION Procedure

This procedure creates a database destination. A database destination represents an Oracle database on which remote database jobs run.

The host that the remote database resides on must have a running Scheduler agent that is registered with the database that this procedure is called from.

Syntax

```
DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION (
    destination_name      IN VARCHAR2,
    agent                 IN VARCHAR2,
    tns_name              IN VARCHAR2,
    comments              IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-23 CREATE_DATABASE_DESTINATION Procedure Parameters

Parameter	Description
destination_name	The name to assign to the database destination. It can optionally be prefixed with a schema name. Cannot be <code>NULL</code> . It is converted to uppercase unless enclosed in double quotation marks.
agent	<p>The external destination name of the Scheduler agent to connect. Equivalent to an agent name.</p> <p>The external destination must already exist. The external destination representing an agent is created automatically on a database instance when the agent registers with that instance.</p> <p>An agent's name is specified in its agent configuration file. If it is not specified, it defaults to the first part (before the first period) of the name of the host it resides on.</p>
tns_name	<p>An Oracle Net connect identifier that is resolved to the Oracle database instance being connected to. The exact syntax depends on the Oracle Net configuration. The connect identifier can be a complete Oracle Net connect descriptor (network address and database service name) or a <i>net service name</i>, which is an alias for a connect descriptor. The alias must be resolved in the <code>tnsnames.ora</code> file on the local computer. The maximum size for <code>tns_name</code> is 2000 characters.</p> <p>If <code>tns_name</code> is <code>NULL</code>, the agent connects to the default Oracle database on its host. You specify the default database by assigning values to the <code>ORACLE_HOME</code> and <code>ORACLE_SID</code> parameters in the agent configuration file, <code>schagent.conf</code>, located in the agent home directory.</p> <p>See <i>Oracle Database Net Services Administrator's Guide</i> for more information on connect identifiers.</p>
comments	A text string that describes the database destination. Scheduler does not use this argument.

Usage Notes

Database destinations reside in a particular schema and can be created by any user with the `CREATE JOB` system privilege. To create a database destination in a schema other than your own, you must have the `CREATE ANY JOB` privilege.

CREATE_EVENT_SCHEDULE Procedure

This procedure creates an event schedule, which is used to start a job when a particular event is raised.

Syntax

```
DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
    schedule_name      IN VARCHAR2,
    start_date         IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    event_condition     IN VARCHAR2 DEFAULT NULL,
    queue_spec         IN VARCHAR2,
    end_date           IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    comments           IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-24 CREATE_EVENT_SCHEDULE Parameters

Parameter	Description
<code>schedule_name</code>	The name to assign to the schedule. The name must be unique in the SQL namespace. For example, a schedule cannot have the same name as a table in a schema. If no name is specified, then an error occurs.
<code>start_date</code>	This attribute specifies the date and time that this schedule becomes valid. Occurrences of the event before this date and time are ignored in the context of this schedule.
<code>event_condition</code>	This is a conditional expression based on the columns of the event source queue table. The expression must have the syntax of an Advanced Queuing rule. Accordingly, you can include user data properties in the expression, provided that the message payload is an object type, and that you prefix object attributes in the expression with <code>tab.user_data</code> . For more information on rules, see the <code>DBMS_AQADM.ADD_SUBSCRIBER</code> procedure.
<code>queue_spec</code>	This argument specifies either a file watcher name or the queue into which events that start this particular job are enqueued (the source queue). If the source queue is a secure queue, the <code>queue_spec</code> argument is a string containing a pair of values of the form <i>queue_name, agent name</i> . For non-secure queues, only the queue name need be provided. If a fully qualified queue name is not provided, the queue is assumed to be in the job owner's schema. In the case of secure queues, the agent name provided should belong to a valid agent that is currently subscribed to the queue.
<code>end_date</code>	The date and time after which jobs do not run and windows do not open. An event schedule that has no <code>end_date</code> is valid forever. <code>end_date</code> must be after the <code>start_date</code> . If it is not, then an error is generated when the schedule is created.
<code>comments</code>	This attribute specifies an optional comment about the schedule. By default, this attribute is <code>NULL</code> .

Usage Notes

You must have the `CREATE JOB` privilege to create a schedule in your own schema or the `CREATE ANY JOB` privilege to create a schedule in someone else's schema by specifying `schema.schedule_name`. Once a schedule has been created, it can be used by other users. The schedule is created with access to `PUBLIC`. Therefore, there is no need to explicitly grant access to the schedule.



See Also:

["CREATE_FILE_WATCHER Procedure"](#)

CREATE_FILE_WATCHER Procedure

This procedure creates a file watcher, which is a Scheduler object that defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job. After you create a file watcher, you reference it in an event-based job or event schedule.

Syntax

```
DBMS_SCHEDULER.CREATE_FILE_WATCHER (
  file_watcher_name      IN VARCHAR2,
  directory_path         IN VARCHAR2,
  file_name              IN VARCHAR2,
  credential_name        IN VARCHAR2,
  destination            IN VARCHAR2 DEFAULT NULL,
  min_file_size          IN PLS_INTEGER DEFAULT 0,
  steady_state_duration  IN INTERVAL DAY TO SECOND DEFAULT NULL,
  comments               IN VARCHAR2 DEFAULT NULL,
  enabled                IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 174-25 CREATE_FILE_WATCHER Parameters

Parameter	Description
<code>file_watcher_name</code>	The name to assign to the file watcher. The name must be unique in the SQL namespace. For example, a file watcher cannot have the same name as a table in a schema. This can optionally be prefixed with a schema name. Cannot be <code>NULL</code> .
<code>directory_path</code>	Directory in which the file is expected to arrive. The single wildcard '?' at the beginning of the path denotes the Oracle home path. For example, '?/rdbms/log' denotes the rdbms/log subdirectory of the Oracle home directory.
<code>file_name</code>	Name of the file to look for. Two wildcards are permitted anywhere in the file name: '?' denotes any single character, and '*' denotes zero or more characters. This attribute cannot be <code>NULL</code> .
<code>credential_name</code>	Name of a valid credential object. The file watcher uses the credential to authenticate itself with the host operating system to access the watched-for file. The file watcher owner must have <code>EXECUTE</code> privileges on the credential. Cannot be <code>NULL</code> .

Table 174-25 (Cont.) CREATE_FILE_WATCHER Parameters

Parameter	Description
destination	Name of an external destination. You create an external destination by registering a remote Scheduler agent with the database. See the view <code>ALL_SCHEDULER_EXTERNAL_DESTS</code> for valid external destination names. If this parameter is <code>NULL</code> , the file watcher is created on the local host.
min_file_size	Minimum size in bytes that the file must be before the file watcher considers the file found. Default is 0.
steady_state_duration	Minimum time interval that the file must remain unchanged before the file watcher considers the file found. Cannot exceed one hour. If <code>NULL</code> , an internal value is used. The minimum value is 10 seconds. Oracle recommends similar <code>steady_state_duration</code> values for all file watchers for efficient file watcher job operation. Also, the repeat interval of the file watcher schedule must be equal or greater than the <code>steady_state_duration</code> value.
comments	Optional comment.
enabled	If <code>TRUE</code> (the default), the file watcher is enabled.

Usage Notes

You must have the `CREATE JOB` system privilege to create a file watcher in your own schema. You require the `CREATE ANY JOB` system privilege to create a file watcher in a schema different from your own (except the `SYS` schema, which is disallowed).

CREATE_GROUP Procedure

This procedure creates a group. Groups contain members, which you can specify when you create the group or at a later time. There are three types of groups: window groups, database destination groups, and external destination groups.

You can use a group name in other `DBMS_SCHEDULER` package procedures to specify a list of objects. For example, to specify multiple destinations for a remote database job, you provide a group name for the `DESTINATION_NAME` parameter of the job.

Syntax

```
DBMS_SCHEDULER.CREATE_GROUP (
    group_name      IN VARCHAR2,
    group_type      IN VARCHAR2,
    member         IN VARCHAR2 DEFAULT NULL,
    comments       IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-26 CREATE_GROUP Procedure Parameters

Parameter	Description
group_name	The name to assign to the group. It can optionally be prefixed with a schema name. It cannot be <code>NULL</code> . It is converted to uppercase unless enclosed in double quotation marks.

Table 174-26 (Cont.) CREATE_GROUP Procedure Parameters

Parameter	Description
group_type	<p>The type of members in the group. All members must be of the same type. Possible types are:</p> <ul style="list-style-type: none"> 'DB_DEST' Database destination: Members are database destinations, for running remote database jobs. 'EXTERNAL_DEST' External destination: Members are external destinations, for running remote external jobs. 'WINDOW' Members are Scheduler windows. You must have the <code>MANAGE SCHEDULER</code> privilege to create a group of this type. <p>Members in database destination and external destination groups have the following format:</p> <pre>[[schema.]credential@][schema.]destination</pre> <p>where:</p> <ul style="list-style-type: none"> <i>credential</i> is the name of an existing credential. <i>destination</i> is the name of an existing database destination or external destination. <p>The credential portion of a destination member is optional. If omitted, the job using this destination member uses its default credential.</p> <p>Members in window groups are window names. Because all Scheduler windows reside in the <code>SYS</code> schema, you do not specify a schema name for windows.</p>
member	<p>Optional comma-separated list of group members. The default is <code>NULL</code>. If <code>NULL</code>, use the <code>ADD_GROUP_MEMBER</code> procedure to add members. You can also use <code>ADD_GROUP_MEMBER</code> to add additional members at a later time.</p> <p>The keyword <code>LOCAL</code> can be used as a member in database destination groups and external destination groups.</p> <ul style="list-style-type: none"> In database destination groups, <code>LOCAL</code> represents the source database on which the job is created. It cannot be preceded with a credential. In external destination groups, <code>LOCAL</code> represents the host on which the source database resides. It can be optionally preceded with a credential name. If no credential is provided, jobs that use this group as their destination must have a default credential.
comments	A text string that describes the group. Scheduler does not use this argument.

Usage Notes

Groups reside in a particular schema and can be created by any user with the `CREATE JOB` system privilege. To create a group in a schema other than your own, you must have the `CREATE ANY JOB` privilege. The group name must be unique among all Scheduler objects.

You can grant the `SELECT` or `READ` privilege on a group so that other users can reference the group when creating jobs or schedules. To enable other users to modify a group, you can grant the `ALTER` privilege on the group.

Each group member must be unique within the group. For destination groups, the credential/destination name pairs must be unique within the group. An error is generated if any of the

group members do not exist. For destination groups, both the credential and destination portions of a member must exist.

Another group of the same type can be a group member. The Scheduler immediately expands the included group name into its list of members.

Groups are created enabled, but you can disable them.

Example

The following PL/SQL block creates a group named `production_dest1`, whose members are database destinations for a collection of production databases.

```
BEGIN
  DBMS_SCHEDULER.CREATE_GROUP (
    GROUP_NAME      => 'production_dest1',
    GROUP_TYPE      => 'DB_DEST',
    MEMBER          => 'LOCAL, oracle_cred@prodhost1, prodhost2',
    COMMENTS        => 'All sector1 production machines');
END;
```

CREATE_INCOMPATIBILITY Procedure

This procedure creates an incompatibility definition.

Syntax

```
DBMS_SCHEDULER.CREATE_INCOMPATIBILITY (
  incompatibility_name  IN VARCHAR2,
  object_name          IN VARCHAR2,
  constraint_level      IN VARCHAR2 DEFAULT 'JOB_LEVEL',
  enabled              IN BOOLEAN DEFAULT TRUE,
  comments              IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-27 CREATE_INCOMPATIBILITY Procedure Parameters

Parameter	Description
<code>incompatibility_name</code>	The name of the incompatibility definition.
<code>object_name</code>	One or more (comma-separated) programs or jobs.
<code>constraint_level</code>	One or more (comma-separated) programs or jobs.
<code>enabled</code>	Specifies whether the constraint is initially enabled (true) or not enabled (false).
<code>comments</code>	Optional descriptive comment.

Usage Notes

If `object_name` contains multiple (comma-separated) values, they must be either all programs or all jobs that are incompatible with each other (that is, they cannot be run at the same time). For jobs, the list must consist of two or more jobs, and `constraint_level` must be 'JOB_LEVEL'. For programs, `constraint_level` can be either 'JOB_LEVEL' or 'PROGRAM_LEVEL'. When set to the default value 'JOB_LEVEL', only a single job that is based on the program (or programs) mentioned in `object_name` can run at the same time. When

`constraint_level` is set to 'PROGRAM_LEVEL', the programs are incompatible, but the jobs based on the same program are not incompatible.

For example, if the value of `object_name` is 'P1,P2,P3' and `constraint_level` is 'PROGRAM_LEVEL', many jobs based on P1 can be running at the same time, but if any P1 based job is running, none based on P2 or P3 can be running. Or, similarly, many jobs based on P3 can be running at the same time, but none based on P1 or P2. If `constraint_level` is set to 'JOB_LEVEL', then only a single job out of all the jobs based on programs P1, P2 and P3 can be running at a time.



See Also:

Using Incompatibility Definitions in *Oracle Database Administrator's Guide*

CREATE_JOB Procedure

This procedure creates a single job.

If you create the job as enabled by setting the `enabled` attribute to `TRUE`, the Scheduler automatically runs the job according to its schedule. If you create the job disabled, the job does not run until you enable it with the [SET_ATTRIBUTE Procedure](#).

The procedure is overloaded. The different functionality of each form of syntax is presented along with the syntax declaration.

Syntax

Creates a job in a single call without using an existing program or schedule:

```
DBMS_SCHEDULER.CREATE_JOB (
    job_name          IN VARCHAR2,
    job_type          IN VARCHAR2,
    job_action        IN VARCHAR2,
    number_of_arguments IN PLS_INTEGER          DEFAULT 0,
    start_date        IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    repeat_interval    IN VARCHAR2              DEFAULT NULL,
    end_date          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    job_class         IN VARCHAR2              DEFAULT 'DEFAULT_JOB_CLASS',
    enabled           IN BOOLEAN                DEFAULT FALSE,
    auto_drop         IN BOOLEAN                DEFAULT TRUE,
    comments          IN VARCHAR2              DEFAULT NULL,
    credential_name   IN VARCHAR2              DEFAULT NULL,
    destination_name  IN VARCHAR2              DEFAULT NULL);
```

Creates a job using a named schedule object and a named program object:

```
DBMS_SCHEDULER.CREATE_JOB (
    job_name          IN VARCHAR2,
    program_name      IN VARCHAR2,
    schedule_name     IN VARCHAR2,
    job_class         IN VARCHAR2              DEFAULT 'DEFAULT_JOB_CLASS',
    enabled           IN BOOLEAN                DEFAULT FALSE,
    auto_drop         IN BOOLEAN                DEFAULT TRUE,
    comments          IN VARCHAR2              DEFAULT NULL,
    job_style         IN VARCHAR2              DEFAULT 'REGULAR',
    credential_name   IN VARCHAR2              DEFAULT NULL,
    destination_name  IN VARCHAR2              DEFAULT NULL);
```

Creates a job using a named program object and an inlined schedule:

```
DBMS_SCHEDULER.CREATE_JOB (
    job_name          IN VARCHAR2,
    program_name      IN VARCHAR2,
    start_date        IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    repeat_interval    IN VARCHAR2                DEFAULT NULL,
    end_date          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    job_class         IN VARCHAR2                DEFAULT 'DEFAULT_JOB_CLASS',
    enabled           IN BOOLEAN                 DEFAULT FALSE,
    auto_drop         IN BOOLEAN                 DEFAULT TRUE,
    comments          IN VARCHAR2                DEFAULT NULL,
    job_style         IN VARCHAR2                DEFAULT 'REGULAR',
    credential_name    IN VARCHAR2                DEFAULT NULL,
    destination_name  IN VARCHAR2                DEFAULT NULL);
```

Creates a job using a named schedule object and an inlined program:

```
DBMS_SCHEDULER.CREATE_JOB (
    job_name          IN VARCHAR2,
    schedule_name     IN VARCHAR2,
    job_type          IN VARCHAR2,
    job_action        IN VARCHAR2,
    number_of_arguments IN PLS_INTEGER           DEFAULT 0,
    job_class         IN VARCHAR2                DEFAULT 'DEFAULT_JOB_CLASS',
    enabled           IN BOOLEAN                 DEFAULT FALSE,
    auto_drop         IN BOOLEAN                 DEFAULT TRUE,
    comments          IN VARCHAR2                DEFAULT NULL,
    credential_name    IN VARCHAR2                DEFAULT NULL,
    destination_name  IN VARCHAR2                DEFAULT NULL);
```

Creates a job using an inlined program and an event:

```
DBMS_SCHEDULER.CREATE_JOB (
    job_name          IN VARCHAR2,
    job_type          IN VARCHAR2,
    job_action        IN VARCHAR2,
    number_of_arguments IN PLS_INTEGER           DEFAULT 0,
    start_date        IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    event_condition    IN VARCHAR2                DEFAULT NULL,
    queue_spec        IN VARCHAR2,
    end_date          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    job_class         IN VARCHAR2                DEFAULT 'DEFAULT_JOB_CLASS',
    enabled           IN BOOLEAN                 DEFAULT FALSE,
    auto_drop         IN BOOLEAN                 DEFAULT TRUE,
    comments          IN VARCHAR2                DEFAULT NULL,
    credential_name    IN VARCHAR2                DEFAULT NULL,
    destination_name  IN VARCHAR2                DEFAULT NULL);
```

Creates a job using a named program object and an event:

```
DBMS_SCHEDULER.CREATE_JOB (
    job_name          IN VARCHAR2,
    program_name      IN VARCHAR2,
    start_date        IN TIMESTAMP WITH TIME ZONE,
    event_condition    IN VARCHAR2,
    queue_spec        IN VARCHAR2,
    end_date          IN TIMESTAMP WITH TIME ZONE,
    job_class         IN VARCHAR2                DEFAULT 'DEFAULT_JOB_CLASS',
    enabled           IN BOOLEAN                 DEFAULT FALSE,
    auto_drop         IN BOOLEAN                 DEFAULT TRUE,
    comments          IN VARCHAR2                DEFAULT NULL,
    job_style         IN VARCHAR2                DEFAULT 'REGULAR',
```

```
credential_name      IN VARCHAR2      DEFAULT NULL,
destination_name     IN VARCHAR2      DEFAULT NULL);
```

Parameters

Table 174-28 *CREATE_JOB Procedure Parameters*

Parameter	Description
job_name	<p>The name to assign to the job. The name must be unique in the SQL namespace. For example, a job cannot have the same name as a table in a schema. If the job being created will reside in another schema, it must be qualified with the schema name.</p> <p>If job_name is not specified, an error is generated. If you want to have a name generated by the Scheduler, you can use the <code>GENERATE_JOB_NAME</code> procedure to generate a name and then use the output in the <code>CREATE_JOB</code> procedure. The <code>GENERATE_JOB_NAME</code> procedure generates a number from a sequence, which is the job name. You can prefix the number with a string. The job name will then be the string with the number from the sequence appended to it. See "GENERATE_JOB_NAME Function" for more information.</p>

Table 174-28 (Cont.) CREATE_JOB Procedure Parameters

Parameter	Description
job_type	<p>This attribute specifies the type of job that you are creating. If it is not specified, an error is generated. See <code>job_action</code> in the next row for related information.</p> <p>The supported values are:</p> <ul style="list-style-type: none"> • 'PLSQL_BLOCK' <p>This specifies that the job is an anonymous PL/SQL block. Job or program arguments are not supported when the job or program type is <code>PLSQL_BLOCK</code>. In this case, the number of arguments must be 0.</p> • 'STORED_PROCEDURE' <p>This specifies that the job is a PL/SQL or Java stored procedure, or an external C subprogram. Only procedures, not functions with return values, are supported.</p> • 'EXECUTABLE' <p>This specifies that the job is going to be run outside the database using an external executable. External jobs are anything that can be executed from the command line of the operating system. Anydata arguments are not supported with a job or program type of <code>EXECUTABLE</code>. The job owner must have the <code>CREATE EXTERNAL JOB</code> system privilege before the job can be enabled or run.</p> • 'CHAIN' <p>This specifies that the job is a chain. Arguments are not supported for a chain, so <code>number_of_arguments</code> must be 0.</p> • 'EXTERNAL_SCRIPT' <p>This specifies that the job is an external script that uses the command shell of the computer running the job. For Windows this is <code>cmd.exe</code> and for UNIX based systems the <code>sh</code> shell, unless a different interpreter is specified by prefixing the first line of the script with <code>#!</code>.</p> • 'SQL_SCRIPT' <p>This specifies that the job is a SQL*Plus script.</p> <p>The job must point to a credential that contains a valid operating system username and password. The SQL*Plus script is run by the SQL*Plus executable. The job may point to a connect credential that contains a database credential. If so, this credential is used to connect to the database before running the SQL*Plus script.</p> <p>Note that if you choose to use connect credential, you must use <code>set_attribute</code> to specify the <code>Connect_Credential_Name</code> attribute. If you do not have connect credential, you must include an explicit SQL*Plus connect statement providing a valid database userid / password.</p> <p>The job owner must have the <code>CREATE EXTERNAL JOB</code> system privilege.</p> • 'BACKUP_SCRIPT' <p>This specifies that the job is an RMAN backup script.</p> <p>The script runs a connect statement that uses either a password or OS authentication before it executes any target commands. The job points to a credential that contains a valid operating system username and password. The RMAN session runs under this operating system user.</p> <p>The Scheduler uses the RMAN executable from the current Oracle home to run the script and throws an error if this is missing.</p> <p>The job owner must have the <code>CREATE EXTERNAL JOB</code> system privilege.</p>

Table 174-28 (Cont.) CREATE_JOB Procedure Parameters

Parameter	Description
job_action	<p>This attribute specifies the action of the job. If job_action is not specified for an inline program, then an error is generated when creating the job.</p> <p>The job action is executed inside an autonomous transaction, and all autonomous transaction guidelines and restrictions apply. For example, online DDL operations are not allowed inside an autonomous transaction, and therefore cannot be used in the job action.</p> <p>The following actions are possible:</p> <ul style="list-style-type: none"> For a PL/SQL block: <p>The action is to execute PL/SQL code. These blocks must end with a semicolon. For example, <code>my_proc();</code> or <code>BEGIN my_proc(); END;</code> or <code>DECLARE arg pls_integer:= 10; BEGIN my_proc2(arg); END;</code>.</p> <p>Note that the Scheduler wraps job_action in its own block and passes the following to PL/SQL for execution: <code>DECLARE ... BEGIN job_action END;</code> This is done to declare some internal Scheduler variables. You can include any Scheduler metadata attribute except event_message in your PL/SQL code. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value.</p> <p>See Table 174-40 for details on available metadata attributes.</p> For a stored procedure: <p>The action is the name of the stored procedure. You have to specify the schema if the procedure resides in another schema than the job. If case sensitivity is needed, enclose the schema name and the store procedure name in double quotes. For example, <code>job_action_action=>'\"Schema\".\"Procedure\"'</code>.</p> <p>PL/SQL procedures with INOUT or OUT arguments are not supported as job_action when the job or program type is STORED_PROCEDURE.</p> For an executable: <p>The action is the name of the external executable, including the full path name, but excluding any command-line arguments. If the action starts with a single question mark ('?'), the question mark is replaced by the path to the Oracle home directory for a local job or to the Scheduler agent home for a remote job. If the action contains an at-sign('@') and the job is local, the at-sign is replaced with the SID of the current Oracle instance.</p> <p>NOTE: Shell script syntax is not supported, only syntax for the name of and path to an executable is supported.</p> For a chain: <p>The action is the name of a Scheduler chain object. You must specify the schema of the chain if it resides in a different schema than the job.</p> For an external script: <p>The job_action must be either the path to an operating system script or an inline operating system script. If the job_action is a path to a script, then the script must reside on every computer that the job runs on. The job_action may contain calls to SQL*Plus or RMAN executables directly, without having to specify its full path, given that they are stored on their default location for every computer that runs the job.</p> <p>The job can only have arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called. The job must point to a credential that contains a valid operating system username and password.</p> For a SQL script:

Table 174-28 (Cont.) CREATE_JOB Procedure Parameters

Parameter	Description
	<p>The <code>job_action</code> must be either the path to a SQL*Plus script or an inline SQL*Plus script. If the <code>job_action</code> is a path to a script, then the script must reside on every computer that the job runs on.</p> <p>The job can only have arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called. If the arguments are named, they are also bound to named variables in the SQL*Plus session.</p> <ul style="list-style-type: none"> For a backup script: <p>The <code>job_action</code> is either the path to a RMAN script or an inline RMAN script. If the <code>program_action</code> is a path to a script, then the script must reside on every computer that the program runs on.</p> <p>The job can only have arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called.</p>
<code>number_of_arguments</code>	This attribute specifies the number of arguments that the job expects. The range is 0-255, with the default being 0.
<code>program_name</code>	The name of the program associated with this job. If the program is of type EXECUTABLE, the job owner must have the CREATE EXTERNAL JOB system privilege before the job can be enabled or run.
<code>start_date</code>	<p>This attribute specifies the first date and time on which this job is scheduled to start. If <code>start_date</code> and <code>repeat_interval</code> are left null, then the job is scheduled to run as soon as the job is enabled.</p> <p>For repeating jobs that use a calendaring expression to specify the repeat interval, <code>start_date</code> is used as a reference date. The first time the job runs is the first match of the calendaring expression that is on or after the current date and time.</p> <p>The Scheduler cannot guarantee that a job executes on an exact time because the system may be overloaded and thus resources unavailable.</p>
<code>event_condition</code>	<p>This is a conditional expression based on the columns of the event source queue table. The expression must have the syntax of an Advanced Queuing rule. Accordingly, you can include user data properties in the expression provided that the message payload is an object type, and that you prefix object attributes in the expression with <code>tab.user_data</code>. For more information on rules, see the DBMS_AQADM.ADD_SUBSCRIBER procedure.</p>
<code>queue_spec</code>	<p>This argument specifies either of the following:</p> <ul style="list-style-type: none"> The source queue where events that start this particular job are enqueued. If it is secure, then the <code>queue_spec</code> argument is a pair of values of the form <code>queue_name, agent_name</code>. If it is not secure, then only the queue name need be provided. If a fully qualified queue name is not provided, the queue is assumed to be in the job owner's schema. In the case of secure queues, the agent name provided should belong to a valid agent that is currently subscribed to the queue. A file watcher name. For more information on this option, see <i>Oracle Database Administrator's Guide</i>.
<code>repeat_interval</code>	<p>This attribute specifies how often the job repeats. You can specify the repeat interval by using calendaring or PL/SQL expressions.</p> <p>The expression specified is evaluated to determine the next time the job should run. If <code>repeat_interval</code> is not specified, the job runs only once at the specified start date. See "Calendaring Syntax" for further information.</p>
<code>schedule_name</code>	The name of the schedule, window, or window group associated with this job.
<code>job_class</code>	The class this job is associated with.

Table 174-28 (Cont.) CREATE_JOB Procedure Parameters

Parameter	Description
end_date	<p>This attribute specifies the date and time after which the job expires and is no longer run. After the end_date, if auto_drop is TRUE, the job is dropped. If auto_drop is FALSE, the job is disabled and the STATE of the job is set to COMPLETED.</p> <p>If no value for end_date is specified, the job repeats forever unless max_runs or max_failures is set, in which case the job stops when either value is reached.</p> <p>The value for end_date must be after the value for start_date. If end_date is less than start_date, then an error will be generated. If end_date is the same as start_date, then the job will not execute and no error will be generated.</p>
comments	This attribute specifies a comment about the job. By default, this attribute is NULL.
job_style	<p>Style of the job being created. This argument can have one of the following values:</p> <ul style="list-style-type: none"> 'REGULAR' creates a regular job. This is the default. 'LIGHTWEIGHT' creates a lightweight job. This value is permitted only when the job references a program object. Use lightweight jobs when you have many short-duration jobs that run frequently. Under certain circumstances, using lightweight jobs can deliver a small performance gain. 'IN_MEMORY_RUNTIME' creates an in-memory runtime job. These jobs are based on lightweight job structures, so the same rules and restrictions apply; however, they further boost performance by keeping an in-memory cache, so they minimize disk access for pre-run and post-run actions. 'IN_MEMORY_FULL' creates an in-memory full job. In-memory full jobs require a program and cannot have a schedule or repeat interval. They run automatically when the job is enabled, and after running they are discarded. They keep all the job information in memory and are not backed up on disk, meaning that they are lost when the instance is rebooted. They are designed to run actions that must be performed immediately with the least amount of overhead possible.
credential_name	<p>The default credential to use with the job. Applicable only to remote database jobs, remote external jobs, local external jobs, script jobs, and event-based jobs that process file arrival events. The credential must exist.</p> <p>For local database jobs, it must be NULL.</p> <p>For local external jobs only, if this attribute is NULL (the default), then a preferred (default) credential is selected. See <i>Oracle Database Administrator's Guide</i> for information about preferred credentials for local external jobs.</p> <p>See also: "CREATE_CREDENTIAL Procedure"</p>

Table 174-28 (Cont.) CREATE_JOB Procedure Parameters

Parameter	Description
destination_name	<p>The database destination or external destination for the job. Use for remote database jobs and remote external jobs only. Must be NULL for jobs running on the local database or for local external jobs (executables).</p> <p>This attribute can be a single destination name or the name of a group of type 'EXTERNAL_DEST' or 'DB_DEST'. The single destination or group must already exist.</p> <p>The following applies to this attribute:</p> <ul style="list-style-type: none"> If it is a database destination, it must have been created by the CREATE_DATABASE_DESTINATION Procedure. If it is an external destination, it must have been implicitly created by registering a remote Scheduler agent with the local database. If it is a group, each member of the group must exist, and the job must run on all destinations named in the group. See "CREATE_GROUP Procedure". <p>destination_name cannot reference a destination group when:</p> <ul style="list-style-type: none"> The job type is 'CHAIN' The job style is 'LIGHTWEIGHT', 'IN_MEMORY_RUNTIME', or 'IN_MEMORY_FULL', <p>If the credential_name argument of CREATE_JOB is NULL, each destination must be preceded by a credential, in the following format:</p> <p><i>credential.destination</i></p> <p>The credential must already exist. If the credential_name argument is provided, then it serves as the default credential for every destination that is not preceded by a credential.</p> <p>You can query the views *_SCHEDULER_DB_DESTS and ALL_SCHEDULER_EXTERNAL_DESTS for existing destinations and *_SCHEDULER_GROUP_MEMBERS for existing groups and their members.</p> <p>*** destination job attribute is deprecated in Oracle Database 11gR2 and superseded by destination_name.</p>
enabled	<p>This attribute specifies whether the job is created enabled or not. The possible settings are TRUE or FALSE. By default, this attribute is set to FALSE and, therefore, the job is created as disabled. A disabled job means that the metadata about the job has been captured, and the job exists as a database object. However, the Scheduler ignores the job and the job coordinator does not pick it for processing. In order for the job coordinator to process the job, the job must be enabled. You can enable a job by setting this argument to TRUE or by using the ENABLE procedure.</p>
auto_drop	<p>This flag, if TRUE, causes a job to be automatically dropped after it has completed or has been automatically disabled. A job is considered completed if:</p> <ul style="list-style-type: none"> Its end date (or the end date of the job schedule) has passed. Note that a job with a Window schedule will not be auto-dropped when the window closes, because this is not considered to be the end of the Window. It has run max_runs number of times. max_runs must be set with SET_ATTRIBUTE. It is not a repeating job and has run once. <p>A job is disabled when it has failed max_failures times. max_failures is also set with SET_ATTRIBUTE.</p> <p>If this flag is set to FALSE, the jobs are not dropped and their metadata is kept until the job is explicitly dropped with the DROP_JOB procedure.</p> <p>By default, jobs are created with auto_drop set to TRUE.</p>

Usage Notes

Jobs are created as disabled by default. You must explicitly enable them so that they will become active and scheduled. Before enabling a job, ensure that all program arguments, if any, are defined, either by defining default values in the program object or by supplying values with the job.

The `JOB_QUEUE_PROCESSES` initialization parameter specifies the maximum number of processes that can be created for the execution of jobs. Beginning with Oracle Database 11g Release 2, `JOB_QUEUE_PROCESSES` applies to `DBMS_SCHEDULER` jobs. Setting this parameter to 0 disables `DBMS_SCHEDULER` jobs.

To create a job in your own schema, you need to have the `CREATE JOB` privilege. A user with the `CREATE ANY JOB` privilege can create a job in any schema. If the job being created will reside in another schema, the job name must be qualified with the schema name. For a job of type `EXECUTABLE` (or for a job that points to a program of type `EXECUTABLE`), the job owner must have the `CREATE EXTERNAL JOB` system privilege before the job can be enabled or run.

Associating a job with a particular class or program requires `EXECUTE` privileges for that class or program.

Not all possible job attributes can be set with `CREATE_JOB`. Some must be set after the job is created. For example, job arguments must be set with the [SET_JOB_ARGUMENT_VALUE Procedure](#) or the [SET_JOB_ANYDATA_VALUE Procedure](#). Other job attributes, such as `job_priority` and `max_runs`, are set with the [SET_ATTRIBUTE Procedure](#).

To create multiple jobs efficiently, use the `CREATE_JOBS` procedure.



Note:

The Scheduler runs event-based jobs for each occurrence of an event that matches the event condition of the job. However, events that occur while the job is already running are ignored; the event gets consumed, but does not trigger another run of the job.

CREATE_JOB_CLASS Procedure

This procedure creates a job class. Job classes are created in the `SYS` schema.

Syntax

```
DBMS_SCHEDULER.CREATE_JOB_CLASS (
    job_class_name          IN VARCHAR2,
    resource_consumer_group IN VARCHAR2 DEFAULT NULL,
    service                 IN VARCHAR2 DEFAULT NULL,
    logging_level           IN PLS_INTEGER
                           DEFAULT DBMS_SCHEDULER.LOGGING_RUNS,
    log_history             IN PLS_INTEGER DEFAULT NULL,
    comments                IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-29 CREATE_JOB_CLASS Procedure Parameters

Parameter	Description
<code>job_class_name</code>	<p>The name to assign to the job class. Job classes can only be created in the <code>SYS</code> schema.</p> <p>This attribute specifies the name of the job class and uniquely identifies the job class. The name must be unique in the SQL namespace. For example, a job class cannot have the same name as a table in a schema.</p>
<code>resource_consumer_group</code>	<p>This attribute specifies the resource consumer group that this class is associated with. A resource consumer group is a set of synchronous or asynchronous sessions that are grouped together based on their processing needs. A job class has a many-to-one relationship with a resource consumer group. The resource consumer group that the job class associates with determines the resources that are allocated to the job class.</p> <p>If a resource consumer group is dropped, job classes associated with it are then associated with the default resource consumer group.</p> <p>If no resource consumer group is specified, job classes are associated with the default resource consumer group.</p> <p>If the specified resource consumer group does not exist when creating the job class, an error occurs.</p>
<code>service</code>	<p>This attribute specifies the database service that the jobs in this class have affinity to. In an Oracle RAC environment, this means that the jobs in this class only run on those database instances that are assigned to the specific service.</p> <p>Note that a service can be mapped to a resource consumer group, so you can also control resources allocated to jobs by specifying a service. See <code>DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING</code> for details. If both the <code>resource_consumer_group</code> and <code>service</code> attributes are specified, and if the service is mapped to a resource consumer group, the <code>resource_consumer_group</code> attribute takes precedence.</p> <p>If no service is specified, the job class belongs to the default service, which means it has no service affinity and any one of the database instances within the cluster might run the job. If the service that a job class belongs to is dropped, the job class will then belong to the default service.</p> <p>If the specified service does not exist when creating the job class, then an error occurs.</p>
<code>logging_level</code>	<p>This attribute specifies how much information is logged. The possible options are:</p> <ul style="list-style-type: none"> <code>DBMS_SCHEDULER.LOGGING_OFF</code> No logging is performed for any jobs in this class. <code>DBMS_SCHEDULER.LOGGING_RUNS</code> The Scheduler writes detailed information to the job log for all runs of each job in this class. This is the default. <code>DBMS_SCHEDULER.LOGGING_FAILED_RUNS</code> The Scheduler logs only jobs that failed in this class. <code>DBMS_SCHEDULER.LOGGING_FULL</code> In addition to recording every run of a job, the Scheduler records all operations performed on all jobs in this class. Every time a job is created, enabled, disabled, altered (with <code>SET_ATTRIBUTE</code>), stopped, and so, an entry is recorded in the log.

Table 174-29 (Cont.) CREATE_JOB_CLASS Procedure Parameters

Parameter	Description
log_history	This attribute controls the number of days that job log entries for jobs in this class are retained. It helps prevent the job log from growing indiscriminately. The range of valid values is 0 through 1000000. If set to 0, no history is kept. If NULL (the default), retention days are set by the log_history Scheduler attribute (set with SET_SCHEDULER_ATTRIBUTE).
comments	This attribute is for an optional comment about the job class. By default, this attribute is NULL.

Usage Notes

For users to create jobs that belong to a job class, the job owner must have `EXECUTE` privileges on the job class. Therefore, after the job class has been created, `EXECUTE` privileges must be granted on the job class so that users create jobs belonging to that class. You can also grant the `EXECUTE` privilege to a role.

Creating a job class requires the `MANAGE SCHEDULER` system privilege.

CREATE_JOBS Procedure

This procedure creates multiple jobs and sets the values of their arguments in a single call.

Syntax

```
DBMS_SCHEDULER.CREATE_JOBS (
    jobdef_array      IN SYS.JOB_DEFINITION_ARRAY,
    commit_semantics  IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters**Table 174-30 CREATE_JOBS Procedure Parameters**

Parameter	Description
jobdef_array	The array of job definitions. See "Data Structures" for a description of the <code>JOB_DEFINITION_ARRAY</code> and <code>JOB_DEFINITION</code> datatypes.
commit_semantics	The commit semantics. The following types are supported: <ul style="list-style-type: none"> <code>STOP_ON_FIRST_ERROR</code> returns on the first error. Previous successfully created jobs are committed to disk. This is the default. <code>TRANSACTIONAL</code> returns on the first error and everything that happened before that error is rolled back. <code>ABSORB_ERRORS</code> tries to absorb any errors and attempts to create the rest of the jobs on the list. It commits all successfully created jobs. If errors occur, you can query the view <code>SCHEDULER_BATCH_ERRORS</code> for details.

Usage Notes

This procedure creates many jobs in the context of a single transaction. To realize the desired performance gains, the jobs being created must be grouped in batches of sufficient size. Calling `CREATE_JOBS` with a small array size may not be much faster than calling `CREATE_JOB` once for each job.

You cannot use this procedure to create multiple-destination jobs. That is, the `destination` attribute of the `job_definition` object cannot reference a destination group.

Examples

See *Oracle Database Administrator's Guide*.

CREATE_PROGRAM Procedure

This procedure creates a program.

Syntax

```
DBMS_SCHEDULER.CREATE_PROGRAM (  
    program_name          IN VARCHAR2,  
    program_type          IN VARCHAR2,  
    program_action        IN VARCHAR2,  
    number_of_arguments  IN PLS_INTEGER DEFAULT 0,  
    enabled               IN BOOLEAN DEFAULT FALSE,  
    comments              IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-31 CREATE_PROGRAM Procedure Parameters

Parameter	Description
<code>program_name</code>	The name to assign to the program. The name must be unique in the SQL namespace. For example, a program cannot have the same name as a table in a schema. If no name is specified, then an error occurs.

Table 174-31 (Cont.) CREATE_PROGRAM Procedure Parameters

Parameter	Description
program_type	<p>This attribute specifies the type of program you are creating. If it is not specified then you get an error. These are the supported values for program_type:</p> <ul style="list-style-type: none"> • 'PLSQL_BLOCK' <p>This specifies that the program is a PL/SQL block. Job or program arguments are not supported when the job or program type is PLSQL_BLOCK. In this case, the number of arguments must be 0.</p> • 'STORED_PROCEDURE' <p>This specifies that the program is a PL/SQL or Java stored procedure, or an external C subprogram. Only procedures, not functions with return values, are supported. PL/SQL procedures with INOUT or OUT arguments are not supported.</p> • 'EXECUTABLE' <p>This specifies that the job is going to be run outside the database using an external executable. External programs imply anything that can be executed from the operating system command line. AnyData arguments are not supported with job or program type EXECUTABLE.</p> • 'EXTERNAL_SCRIPT' <p>This specifies that the job is an external script that uses the command shell of the computer running the job. For Windows this is cmd.exe and for UNIX based systems the sh shell, unless a different interpreter is specified by prefixing the first line of the script with #!.</p> • 'SQL_SCRIPT' <p>This specifies that the program is a SQL*Plus script.</p> <p>A job using this program must point to a credential that contains a valid operating system username and password. The SQL*Plus script is run by SQL*Plus executable. The job using this program may point to a connect credential that contains a database credential. If so, this credential is used to connect to the database before running the SQL*Plus script.</p> <p>Note that if you choose to use connect credential, you must use set_attribute to specify the Connect_Credential_Name attribute. If you do not have connect credential, you must include an explicit SQL*Plus connect statement providing a valid database userid / password.</p> • 'BACKUP_SCRIPT' <p>This specifies that the program is an RMAN backup script.</p> <p>The script runs a connect statement that uses either a password or OS authentication before it executes any target commands. The Scheduler uses the RMAN executable from the current Oracle home to run the script and throws an error if this is missing.</p>

Table 174-31 (Cont.) CREATE_PROGRAM Procedure Parameters

Parameter	Description
program_action	<p>This attribute specifies the action of the program. If program_action is not specified, an error is generated.</p> <p>The following actions are possible:</p> <ul style="list-style-type: none"> For a PL/SQL block, the action is to execute PL/SQL code. These blocks must end with a semicolon. For example, <code>my_proc();</code> or <code>BEGIN my_proc(); END;</code> or <code>DECLARE arg pls_integer:= 10; BEGIN my_proc2(arg); END;</code>. Note that the Scheduler wraps job_action in its own block and passes the following to PL/SQL for execution: <code>DECLARE ... BEGIN job_action END;</code> This is done to declare some internal Scheduler variables. You can include any Scheduler metadata attribute except event_message in your PL/SQL code. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value. See Table 174-40 for details on available metadata attributes. If it is an anonymous block, special Scheduler metadata may be accessed using the following variable names: job_name, job_owner, job_start, window_start, window_end. For more information, see the "DEFINE_METADATA_ARGUMENT Procedure". For a stored procedure, the action is the name of the stored procedure. You have to specify the schema if the procedure resides in a schema other than the job. If case sensitivity is needed, enclose the schema name and the store procedure name in double quotes. For example, <code>program_action=>' "Schema"."Procedure" '.</code> For an executable, the action is the name of the external executable, including the full path name, but excluding any command-line arguments. If the action starts with a single question mark ('?'), the question mark is replaced by the path to the Oracle home directory for a local job or to the Scheduler agent home for a remote job. If the action contains an at sign ('@') and the job is local, the at sign is replaced with the SID of the current Oracle instance. For an external script, the action must be either the path to an operating system script or an inline operating system script. If the program_action is a path to a script, then the script must reside on every computer that the program runs on. The program_action may contain calls to SQL*Plus or RMAN executables directly, without having to specify its full path, given that they are stored on their default location for every computer that runs the job. The job can only have arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called. The program points to a credential that contains a valid operating system username and password. For a SQL script, the action must be either the path to a SQL*Plus script or an inline SQL*Plus script. If the program_action is a path to a script, then the script must reside on every computer that the program runs on. The job can only have arguments that are strings or that can be cast to strings. These arguments are passed positionally when

Table 174-31 (Cont.) CREATE_PROGRAM Procedure Parameters

Parameter	Description
	<p>the script is called. If the arguments are named, they are also bound to named variables in the SQL*Plus session.</p> <ul style="list-style-type: none">For a backup script, the action must be either the path to a RMAN script or an inline RMAN script. If the <code>program_action</code> is a path to a script, then the script must reside on every computer that the program runs on. <p>The job can only have arguments that are strings or that can be cast to strings. These arguments are passed positionally when the script is called.</p>
<code>number_of_arguments</code>	<p>This attribute specifies the number of arguments the program takes. If this parameter is not specified, then the default is 0. A program can have a maximum of 255 arguments.</p> <p>If the <code>program_type</code> is <code>PLSQL_BLOCK</code>, then this parameter is ignored.</p>
<code>enabled</code>	<p>This flag specifies whether the program should be created as enabled or not. If the flag is set to <code>TRUE</code>, then validity checks are made and the program is created as <code>ENABLED</code> if all the checks be successful. By default, this flag is set to <code>FALSE</code>, meaning not created enabled. You can also call the <code>ENABLE</code> procedure to enable the program before it can be used.</p>
<code>comments</code>	<p>A comment about the program. By default, this attribute is <code>NULL</code>.</p>

Usage Notes

To create a program in their own schema, users need the `CREATE JOB` privilege. A user with the `CREATE ANY JOB` privilege can create a program in any schema. A program is created in a disabled state by default (unless the enabled parameter is set to `TRUE`). It cannot be executed by a job until it is enabled.

To use your programs, other users must have `EXECUTE` privileges, therefore once a program has been created, you have to grant `EXECUTE` privileges on it.



See Also:

["DEFINE_PROGRAM_ARGUMENT Procedure"](#)

CREATE_RESOURCE Procedure

This procedure allows users to specify the resources used by jobs or to create a new resource.

Syntax

```
DBMS_SCHEDULER.CREATE_RESOURCE (  
  resource_name  IN VARCHAR2,  
  units          IN PLS_INTEGER,  
  status         IN VARCHAR2 DEFAULT 'ENFORCE_CONSTRAINTS',  
  constraint_level IN VARCHAR2 DEFAULT 'JOB_LEVEL',  
  comments       IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-32 CREATE_RESOURCE Procedure Parameters

Parameter	Description
resource_name	The name of the resource.
units	The number of units of this resource that the job or program uses.
status	The status of the resource. <ul style="list-style-type: none">• 'ENFORCE_CONSTRAINTS'. This is the default value, and when set, will force the scheduler to enforce resource limits. When the maximum number of units of this resource has been reached no additional jobs using this resource will get started.• 'IGNORE_CONSTRAINTS'. When set, the scheduler will ignore any constraints on this resource.• 'BLOCKED_ALL_JOBS'. No jobs having a constraint on this resource will be allowed to run. The resource is considered to be permanently blocking until switched to one of the other two states.
constraint_level	Level of the constraint: <code>JOB_LEVEL</code> or <code>PROGRAM_LEVEL</code> . For incompatibilities, for <code>JOB_LEVEL</code> , the incompatibility members must be jobs; for <code>PROGRAM_LEVEL</code> the incompatibility members must be programs.
comments	Descriptive comment about the resource.

Usage Notes

The following example creates a new resource.

```
BEGIN
  DBMS_SCHEDULER.CREATE_RESOURCE(
    resource_name => 'my_resource',
    units => 3,
    state => 'ENFORCE_CONSTRAINTS',
    comments => 'Resource1'
  )
END;
/
```



See Also:

- Creating or Dropping a Resource in *Oracle Database Administrator's Guide*
- [SET_RESOURCE_CONSTRAINT Procedure](#)

CREATE_SCHEDULE Procedure

This procedure creates a schedule.

Syntax

```
DBMS_SCHEDULER.CREATE_SCHEDULE (
  schedule_name          IN VARCHAR2,
```

```

start_date          IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,
repeat_interval     IN VARCHAR2,
end_date            IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,
comments            IN VARCHAR2 DEFAULT NULL);

```

Parameters

Table 174-33 CREATE_SCHEDULE Procedure Parameters

Parameter	Description
schedule_name	The name to assign to the schedule. The name must be unique in the SQL namespace. For example, a schedule cannot have the same name as a table in a schema. If no name is specified, then an error occurs.
start_date	<p>This attribute specifies the first date and time on which this schedule becomes valid. For a repeating schedule, the value for <code>start_date</code> is a reference date. In this case, the start of the schedule is not the <code>start_date</code>; it depends on the repeat interval specified. <code>start_date</code> is used to determine the first instance of the schedule.</p> <p>If <code>start_date</code> is specified in the past and no value for <code>repeat_interval</code> is specified, the schedule is invalid. For a repeating job or window, <code>start_date</code> can be derived from the <code>repeat_interval</code> if it is not specified.</p> <p>If <code>start_date</code> is null, then the date that the job or window is enabled is used. <code>start_date</code> and <code>repeat_interval</code> cannot both be null.</p>
repeat_interval	This attribute specifies how often the schedule repeats. It is expressed using calendaring syntax. See " Calendaring Syntax " for further information. PL/SQL expressions are not allowed as repeat intervals for named schedules.
end_date	<p>The date and time after which jobs will not run and windows will not open.</p> <p>A non-repeating schedule that has no <code>end_date</code> is valid forever.</p> <p><code>end_date</code> has to be after the <code>start_date</code>. If this is not the case, then an error is generated when the schedule is created.</p>
comments	This attribute specifies an optional comment about the schedule. By default, this attribute is NULL.

Usage Notes

This procedure requires the `CREATE JOB` privilege to create a schedule in your own schema or the `CREATE ANY JOB` privilege to create a schedule in someone else's schema by specifying `schema.schedule_name`. Once a schedule has been created, it can be used by other users. The schedule is created with access to `PUBLIC`. Therefore, there is no need to explicitly grant access to the schedule.

CREATE_WINDOW Procedure

This procedure creates a recurring time window and associates it with a resource plan. You can then use the window to schedule jobs that run under the associated resource plan. Windows are created in the `SYS` schema.

The procedure is overloaded.

Syntax

Creates a window using a named schedule object:

```

DBMS_SCHEDULER.CREATE_WINDOW (
    window_name          IN VARCHAR2,

```

```
resource_plan      IN VARCHAR2,
schedule_name     IN VARCHAR2,
duration          IN INTERVAL DAY TO SECOND,
window_priority   IN VARCHAR2 DEFAULT 'LOW',
comments          IN VARCHAR2 DEFAULT NULL);
```

Creates a window using an inlined schedule:

```
DBMS_SCHEDULER.CREATE_WINDOW (
  window_name      IN VARCHAR2,
  resource_plan    IN VARCHAR2,
  start_date       IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  repeat_interval  IN VARCHAR2,
  end_date         IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  duration         IN INTERVAL DAY TO SECOND,
  window_priority  IN VARCHAR2 DEFAULT 'LOW',
  comments         IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-34 CREATE_WINDOW Procedure Parameters

Parameter	Description
window_name	The name to assign to the window. The name must be unique in the SQL namespace. All windows are in the SYS schema, so the preface 'SYS' is optional.
resource_plan	<p>This attribute specifies the resource plan that automatically activates when the window opens. When the window closes, the system switches to the appropriate resource plan, which is usually the plan that was in effect before the window opened, but can also be the plan of a different window.</p> <p>Only one resource plan can be associated with a window. It may be NULL or the empty string (""). When it is NULL, the resource plan in effect when the window opens stays in effect for the duration of the window. When it is the empty string, the resource manager is disabled for the duration of the window.</p> <p>If the window is open and the resource plan is dropped, then the resource allocation for the duration of the window is not affected.</p>
start_date	<p>This attribute specifies the first date and time on which this window is scheduled to open. If the value for start_date specified is in the past or is not specified, the window opens as soon as it is created.</p> <p>For repeating windows that use a calendaring expression to specify the repeat interval, the value for start_date is a reference date. The first time the window opens depends on the repeat interval specified and the value for start_date.</p>
duration	This attribute specifies how long the window stays open. For example, 'interval '5' hour' for five hours. There is no default value for this attribute. Therefore, if no value is specified when the window is created, an error occurs. The duration is of type interval day to seconds and ranges from one minute to 99 days.
schedule_name	This attribute specifies the name of the schedule associated with the window.

Table 174-34 (Cont.) CREATE_WINDOW Procedure Parameters

Parameter	Description
repeat_interval	<p>This attribute specifies how often the window repeats. It is expressed using the Scheduler calendaring syntax. See "Calendaring Syntax" for more information.</p> <p>A PL/SQL expression cannot be used to specify the repeat interval for a window.</p> <p>The expression specified is evaluated to determine the next time the window opens. If no repeat_interval is specified, the window opens only once at the specified start date.</p>
end_date	<p>This attribute specifies the date and time after which the window no longer opens. When the value for end_date is reached, the window is disabled. In the *_SCHEDULER_WINDOWS views, the enabled flag of the window is set to FALSE.</p> <p>A non-repeating window that has no value for end_date opens only once for the duration of the window. For a repeating window, if no end_date is specified, then the window keeps repeating forever.</p> <p>The end_date must be after the start_date. If it is not, then an error is generated when the window is created.</p>
window_priority	<p>This attribute is only relevant when two windows overlap. Because only one window can be in effect at one time, the window priority determines which window opens. The two possible values for this attribute are 'HIGH' and 'LOW'. A high priority window has precedence over a low priority window, therefore, the low priority window does not open if it overlaps a high priority window. By default, windows are created with priority 'LOW'.</p>
comments	<p>This attribute specifies an optional comment about the window. By default, this attribute is NULL.</p>

Usage Notes

Creating a window requires the `MANAGE SCHEDULER` privilege.

Scheduler windows are the principal mechanism used to automatically switch resource plans according to a schedule. You can also manually activate a resource plan by using the `ALTER SYSTEM SET RESOURCE_MANAGER_PLAN` statement or the `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` package procedure. Note that either of these manual methods can also disable resource plan switching by Scheduler windows. For more information, see *Oracle Database Administrator's Guide* and ["SWITCH_PLAN Procedure"](#).

DEFINE_ANYDATA_ARGUMENT Procedure

This procedure defines a name or default value for a program argument that is of a complex type and must be encapsulated within an `ANYDATA` object. A job that references the program can override the default value.

Syntax

```
DBMS_SCHEDULER.DEFINE_ANYDATA_ARGUMENT (
  program_name          IN VARCHAR2,
  argument_position     IN PLS_INTEGER,
  argument_name         IN VARCHAR2 DEFAULT NULL,
  argument_type         IN VARCHAR2,
```

```
default_value      IN SYS.ANYDATA,  
out_argument      IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-35 DEFINE_ANYDATA_ARGUMENT Procedure Parameters

Parameter	Description
program_name	The name of the program to be altered. A program with this name must exist.
argument_position	The position of the argument as it is passed to the executable. Argument numbers go from one to the <code>number_of_arguments</code> specified for the program. This must be unique, so it can replace any argument already defined at this position.
argument_name	The name to assign to the argument. It is optional, but must be unique for the program if it is specified. If you assign a name, the name can then be used by other package procedures, including the SET_JOB_ANYDATA_VALUE Procedure .
argument_type	The datatype of the argument being defined. This is not verified or used by the Scheduler. It is only used by the user of the program when deciding what value to assign to the argument.
default_value	The default value to be assigned to the argument encapsulated within an <code>AnyData</code> object. This is optional.
out_argument	This parameter is reserved for future use. It must be set to <code>FALSE</code> .

Usage Notes

All program arguments from one to the `number_of_arguments` value must be defined before a program can be enabled. If a default value for an argument is not defined with this procedure, a value must be defined in the job.

Defining a program argument requires that you be the owner of the program or have `ALTER` privileges on that program. You can also define a program argument if you have the `CREATE ANY JOB` privilege.

See Also:

- ["DEFINE_PROGRAM_ARGUMENT Procedure"](#)
- ["SET_JOB_ANYDATA_VALUE Procedure"](#)

DEFINE_CHAIN_EVENT_STEP Procedure

This procedure adds or replaces a chain step and associates it with an event schedule or an inline event.

Once started in a running chain, this step does not complete until the specified event has occurred. Every step in a chain must be defined before the chain can be enabled and used. Defining a step gives it a name and specifies what happens during the step. If a step already exists with this name, the new step replaces the old one.

Syntax

```
DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (  
    chain_name          IN VARCHAR2,  
    step_name           IN VARCHAR2,  
    event_schedule_name IN VARCHAR2,  
    timeout              IN INTERVAL DAY TO SECOND DEFAULT NULL);
```

```
DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (  
    chain_name          IN VARCHAR2,  
    step_name           IN VARCHAR2,  
    event_condition     IN VARCHAR2,  
    queue_spec          IN VARCHAR2,  
    timeout              IN INTERVAL DAY TO SECOND DEFAULT NULL);
```

Parameters

Table 174-36 DEFINE_CHAIN_EVENT_STEP Procedure Parameters

Parameter	Description
chain_name	The name of the chain that the step is in
step_name	The name of the step
event_schedule_name	The name of the event schedule that the step waits for
timeout	This parameter is reserved for future use
event_condition	See the CREATE_EVENT_SCHEDULE Procedure
queue_spec	See the CREATE_EVENT_SCHEDULE Procedure

Usage Notes

Defining a chain step requires `ALTER` privileges on the chain either as the owner of the chain, or as a user with the `ALTER` object privilege on the chain or the `CREATE ANY JOB` system privilege.

You can base a chain step on a file watcher as well. To do this, provide the file watcher name directly in the `queue_spec` parameter, or use a file watcher schedule for the `event_schedule_name` parameter.



See Also:

["DEFINE_CHAIN_STEP Procedure"](#)

DEFINE_CHAIN_RULE Procedure

This procedure adds a new rule to an existing chain, specified as a condition-action pair. The condition is expressed using either SQL or the Scheduler chain condition syntax and indicates the prerequisites for the action to occur. The action is a result of the condition being met.

An actual rule object is created to store the rule in the schema where the chain resides. If a rule name is given, this name is used for the rule object. If an existing rule name in the schema of the chain is given, the existing rule is altered. (A schema different than the schema of the chain cannot be specified). If no rule name is given, one is generated in the form `SCHED_RULE${N}`.

Syntax

```
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
    chain_name          IN VARCHAR2,
    condition            IN VARCHAR2,
    action               IN VARCHAR2,
    rule_name           IN VARCHAR2 DEFAULT NULL,
    comments             IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-37 DEFINE_CHAIN_RULE Procedure Parameters

Parameter	Description
chain_name	The name of the chain to alter
condition	<p>A boolean expression which must evaluate to <code>TRUE</code> for the action to be performed. Every chain must have a rule that evaluates to <code>TRUE</code> to start the chain. For this purpose, you can use a rule that has 'TRUE' as its condition if you are using Scheduler chain condition syntax, or '1=1' as its condition if you are using SQL syntax.</p> <ul style="list-style-type: none"> Scheduler Chain Condition Syntax See "Scheduler Chain Condition Syntax" and <i>Oracle Database Administrator's Guide</i> for details SQL WHERE Clause Syntax Conditions expressed with SQL must use the syntax of a <code>SELECT</code> statement <code>WHERE</code> clause. You can refer to chain step attributes by using the chain step name as a bind variable. The bind variable syntax is <code>:step_name.attribute</code>. (<i>step_name</i> refers to a typed object.) Possible attributes are: <code>completed</code>, <code>state</code>, <code>start_date</code>, <code>end_date</code>, <code>error_code</code>, and <code>duration</code>. Possible values for the <code>state</code> attribute include: 'NOT_STARTED', 'SCHEDULED', 'RUNNING', 'PAUSED', 'STALLED', 'SUCCEEDED', 'FAILED', and 'STOPPED'. If a step is in the state 'SUCCEEDED', 'FAILED', or 'STOPPED', its <code>completed</code> attribute is set to 'TRUE', otherwise <code>completed</code> is 'FALSE'.
action	<p>The action to be performed when the rule evaluates to <code>TRUE</code>. The action must consist of at least one keyword with an optional value and an optional delay clause.</p> <p>Possible actions include:</p> <ul style="list-style-type: none"> [AFTER <i>delay_interval</i>] <code>START step_1[,step_2 ...]</code> <code>STOP step_1[,step_2 ...]</code> <code>END [{end_value step_name.error_code}]</code> <p>At the beginning of the <code>START</code> action, a delay clause can specify a delay interval before performing the action. <i>delay_interval</i> is a formatted datetime interval of the form <code>HH:MM:SS</code>.</p> <p>The <code>END</code> action ends the chain with an error code equal to either the supplied <i>end_value</i> or the error code that <i>step_name</i> completes with. The default error code is 0, indicating a successful chain run.</p>
rule_name	The name of the rule being created. If no <code>rule_name</code> is given, one is generated in the form <code>SCHED_RULE\${N}</code> .

Table 174-37 (Cont.) DEFINE_CHAIN_RULE Procedure Parameters

Parameter	Description
comments	An optional comment describing the rule. This is stored in the rule object created.

Scheduler Chain Condition Syntax

The Scheduler chain condition syntax provides an easy way to construct a condition using the states and error codes of steps in the current chain.

Chain Condition Syntax

The following are the available constructs for Scheduler chain condition syntax, which are all boolean expressions:

```
TRUE
FALSE
stepname [NOT] SUCCEEDED
stepname [NOT] FAILED
stepname [NOT] STOPPED
stepname [NOT] COMPLETED
stepname ERROR_CODE IN (integer, integer, integer ...)
stepname ERROR_CODE NOT IN (integer, integer, integer ...)
stepname ERROR_CODE = integer
stepname ERROR_CODE != integer
stepname ERROR_CODE <> integer
stepname ERROR_CODE > integer
stepname ERROR_CODE >= integer
stepname ERROR_CODE < integer
stepname ERROR_CODE <= integer
```

These boolean operators are available to create more complex conditions:

```
expression AND expression
expression OR expression
NOT (expression)
```

integer can be positive or negative. Parentheses may be used for clarity or to enforce ordering. You must use parentheses with the **NOT** operator.

PL/SQL code that runs as part of a step can set the value of **ERROR_CODE** for that step with the **RAISE_APPLICATION_ERROR** statement.

Usage Notes

Defining a chain rule requires **ALTER** privileges on the chain (either as the owner, or as a user with **ALTER** privileges on the chain or the **CREATE ANY JOB** system privilege).

You must define at least one rule that starts the chain and at least one that ends it. See the section "Adding Rules to a Chain" in *Oracle Database Administrator's Guide* for more information.

Examples

The following are examples of using rule conditions and rule actions.

Rule Conditions Using Scheduler Chain Condition Syntax

```
'step1 completed'
-- satisfied when step step1 has completed. (step1 completed is also TRUE when any
-- of the following are TRUE: step1 succeeded, step1 failed, step1 stopped.)

'step1 succeeded and step2 succeeded'
-- satisfied when steps step1 and step2 have both succeeded

'step1 error_code > 100'
-- satisfied when step step1 has failed with an error_code greater than 100

'step1 error_code IN (1, 3, 5, 7)'
-- satisfied when step step1 has failed with an error_code of 1, 3, 5, or 7
```

Rule Conditions Using SQL Syntax

```
':step1.completed = 'TRUE' AND :step1.end_date >SYSDATE-1/24'
--satisfied when step step1 completed less than an hour ago

':step1.duration > interval '5' minute'
-- satisfied when step step1 has completed and took longer than 5 minutes to complete
```

Rule Actions

```
'AFTER 01:00:00 START step1, step2'
--After an hour start steps step1 and step2

'STOP step1'
--Stop step step1

END step4.error_code'
--End the chain with the error code that step step4 finished with. If step4 has not
completed, the chain will be ended unsuccessfully with error code 27435.

'END' or 'END 0'
--End the chain successfully (with error_code 0)

'END 100'
--End the chain unsuccessfully with error code 100.
```

DEFINE_CHAIN_STEP Procedure

This procedure adds or replaces a chain step and associates it with a program or a nested chain. When the chain step is started, the specified program or chain is run. If a step already exists with the name supplied in the `chain_name` argument, the new step replaces the old one.

The chain owner must have `EXECUTE` privileges on the program or chain associated with the step. Only one program or chain can run during a step.

You cannot set all possible step attributes with this procedure. Use the `ALTER_CHAIN` procedure to set additional chain step attributes, such as `credential_name` and `destination_name`.

Syntax

```
DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
    chain_name          IN VARCHAR2,
    step_name           IN VARCHAR2,
    program_name        IN VARCHAR2);
```

Parameters

Table 174-38 DEFINE_CHAIN_STEP Procedure Parameters

Parameter	Description
chain_name	The name of the chain to alter.
step_name	The name of the step being defined. If a step already exists with this name, the new step replaces the old one.
program_name	The name of a program or chain to run during this step. The chain owner must have EXECUTE privileges on this program or chain.

Usage Notes

Defining a chain step requires ALTER privileges on the chain (either as the owner, or a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).



See Also:

- ["ALTER_CHAIN Procedure"](#)
- ["DEFINE_CHAIN_EVENT_STEP Procedure"](#)

DEFINE_METADATA_ARGUMENT Procedure

This procedure defines a special metadata argument for the program. The Scheduler can pass Scheduler metadata through this argument to your stored procedure or other executable. You cannot set values for jobs using this argument.

Syntax

```
DBMS_SCHEDULER.DEFINE_METADATA_ARGUMENT (  
    program_name          IN VARCHAR2,  
    metadata_attribute    IN VARCHAR2,  
    argument_position      IN PLS_INTEGER,  
    argument_name          IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-39 DEFINE_METADATA_ARGUMENT Procedure Parameters

Parameter	Description
program_name	The name of the program to be altered
metadata_attribute	The metadata to be passed. Valid metadata attributes are: 'job_name', 'job_subname', 'job_owner', 'job_start', 'window_start', 'window_end', and 'event_message'. Table 174-40 describes these attributes in detail.
argument_position	The position of the argument as it is passed to the executable. The position cannot be greater than the number_of_arguments specified for the program. It must be unique, so it replaces any argument already defined at this position.

Table 174-39 (Cont.) DEFINE_METADATA_ARGUMENT Procedure Parameters

Parameter	Description
argument_name	The name to assign to the argument. It is optional, but must be unique for the program if it is specified. If you assign a name, the name can then be used by other package procedures.

Table 174-40 Metadata Attributes

Metadata Attribute	Datatype	Description
job_name	VARCHAR2	Name of the currently running job
job_subname	VARCHAR2	Subname of the currently running job. The name + subname form a unique identifier for a job that is running a chain step. NULL if the job is not part of a chain.
job_owner	VARCHAR2	Owner of the currently running job
job_scheduled_start	TIMESTAMP WITH TIME ZONE	When the currently running job was scheduled to start
job_start	TIMESTAMP WITH TIME ZONE	When the currently running job started
window_start	TIMESTAMP WITH TIME ZONE	If the job was started by a window, the time that the window opened
window_end	TIMESTAMP WITH TIME ZONE	If the job was started by a window, the time that the window is scheduled to close
event_message	(See Description)	For an event-based job, the message content of the event that started the job. The datatype of this attribute depends on the queue used for the event. It has the same type as the USER_DATA column of the queue table. In the case of a file arrival event, event_message is of type SYS.SCHEDULER_FILEWATCHER_RESULT. See "SCHEDULER_FILEWATCHER_RESULT Object Type" .

Usage Notes

Defining a program argument requires that you be the owner of the program or have ALTER privileges on that program. You can also define a program argument if you have the CREATE ANY JOB privilege.

All metadata attributes except event_message can be used in PL/SQL blocks that you enter into the job_action or program_action attributes of jobs or programs, respectively. You use the attribute name as you use any other PL/SQL identifier, and the Scheduler assigns it a value.

DEFINE_PROGRAM_ARGUMENT Procedure

This procedure defines a name or default value for a program argument. If no default value is defined for a program argument, the job that references the program must supply an argument value. (The job can also override a default value.)

This procedure is overloaded.

Syntax

Defines a program argument without a default value:

```
PROCEDURE define_program_argument(
    program_name      IN VARCHAR2,
    argument_position  IN PLS_INTEGER,
    argument_name      IN VARCHAR2 DEFAULT NULL,
    argument_type      IN VARCHAR2,
    out_argument       IN BOOLEAN DEFAULT FALSE);
```

Defines a program argument with a default value:

```
PROCEDURE define_program_argument(
    program_name      IN VARCHAR2,
    argument_position  IN PLS_INTEGER,
    argument_name      IN VARCHAR2 DEFAULT NULL,
    argument_type      IN VARCHAR2,
    default_value      IN VARCHAR2,
    out_argument       IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-41 DEFINE_PROGRAM_ARGUMENT Procedure Parameters

Parameter	Description
program_name	The name of the program to be altered. A program with this name must exist.
argument_position	The position of the argument as it is passed to the executable. Argument numbers go from one to the <code>number_of_arguments</code> specified for the program. This must be unique so it replaces any argument already defined at this position.
argument_name	The name to assign to the argument. It is optional, but must be unique for the program if specified. If you assign a name, the name can then be used by other package procedures, including the SET_JOB_ARGUMENT_VALUE Procedure .
argument_type	The datatype of the argument being defined. This is not verified or used by the Scheduler. The program user uses <code>argument_type</code> when deciding what value to assign to the argument. Any valid SQL datatype is allowed.
default_value	The default value to be assigned to the argument if none is specified by the job.
out_argument	This parameter is reserved for future use. It must be set to <code>FALSE</code> .

Usage Notes

All program arguments from 1 to the `number_of_arguments` value must be defined before a program can be enabled. If a default value for an argument is not defined with this procedure, a value must be defined in the job.

Defining a program argument requires that you be the owner of the program or have `ALTER` privileges on that program. You can also define a program argument if you have the `CREATE ANY JOB` privilege.

`DEFINE_PROGRAM_ARGUMENT` only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.

See Also:

- ["DEFINE_ANYDATA_ARGUMENT Procedure"](#)
- ["SET_JOB_ARGUMENT_VALUE Procedure"](#)

DISABLE Procedure

This procedure disables a program, job, chain, window, database destination, external destination, file watcher, or group. When an object is disabled, its `enabled` attribute is set to `FALSE`.

Syntax

```
DBMS_SCHEDULER.DISABLE (  
    name             IN VARCHAR2,  
    force            IN BOOLEAN DEFAULT FALSE,  
    commit_semantics IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters

Table 174-42 DISABLE Procedure Parameters

Parameter	Description
<code>name</code>	The name of the object being disabled. Can be a comma-delimited list. If a job class name is specified, then all the jobs in the job class are disabled. The job class is not disabled. If a group name is specified, then the group is disabled, but the enabled state of the group members is unaffected.
<code>force</code>	If <code>TRUE</code> , objects are disabled even if other objects depend on them. See the usage notes for more information.
<code>commit_semantics</code>	The commit semantics. The following types are supported: <ul style="list-style-type: none"><code>STOP_ON_FIRST_ERROR</code>: The procedure returns on the first error and the previous disable operations that were successful are committed to disk. This is the default.<code>TRANSACTIONAL</code>: The procedure returns on the first error and everything that happened before that error is rolled back. This type is only supported when disabling a job or a list of jobs. In addition, this type is not supported when <code>force</code> is set to <code>TRUE</code>.<code>ABSORB_ERRORS</code>: The procedure tries to absorb any errors and disable the rest of the jobs and commits all the disable operations that were successful. If errors occur, you can query the view <code>SCHEDULER_BATCH_ERRORS</code> for details. This type is only supported when disabling a job or a list of jobs.

Usage Notes

Windows must be preceded by `SYS`.

Disabling an object that is already disabled does not generate an error.

The purpose of the `force` option is to point out dependencies. No dependent objects are altered.

To run `DISABLE` for a window or a group of type `WINDOW`, you must have the `MANAGE SCHEDULER` privilege.

You can use `DISABLE` with any schema except the `SYS` schema.

Jobs

Disabling a job means that, although the metadata of the job is there, it should not run and the job coordinator will not pick up these jobs for processing. When a job is disabled, its `state` in the job queue is changed to `disabled`.

If `force` is set to `FALSE` and the job is currently running, an error is returned.

If `force` is set to `TRUE`, the job is disabled, but the currently running instance is allowed to finish.

For jobs with multiple destinations, you cannot disable a child job at a specific destination. Instead, you can disable the destination.

Programs

When a program is disabled, the status is changed to disabled. A disabled program implies that, although the metadata is still there, jobs that point to this program cannot run.

If `force` is set to `FALSE`, the program must not be referenced by any job, otherwise an error will occur.

If `force` is set to `TRUE`, those jobs that point to the program will not be disabled, however, they will fail at runtime because their program will not be valid.

Running jobs that point to the program are not affected by the `DISABLE` call and are allowed to continue

No arguments that pertain to the program are affected when the program is disabled.

File Watchers

If `force` is set to `FALSE`, the file watcher must not be referenced by any job, otherwise an error will occur. If you force disabling a file watcher, jobs that depend on it become disabled.

Windows

This means that the window will not open, however, the metadata of the window is still there, so it can be reenabled.

If `force` is set to `FALSE`, the window must not be open or referenced by any job otherwise an error occurs.

If `force` is set to `TRUE`, disabling a window that is open will succeed but the window will not be closed. It will prevent the window from opening in the future until it is reenabled.

When the window is disabled, those jobs that have the window as their schedule will not be disabled.

Window Groups

When a group of type `WINDOW` is disabled, jobs (other than a running job) that have the window group as their schedule will not run when the member windows open. However, a job that has one of the window group members as its schedule still runs.

The metadata of the window group is still there, so it can be reenabled. Note that the members of the window group will still open.

If `force` is set to `FALSE`, the window group must not have any members that are open or referenced by any job, otherwise an error will occur.

If `force` is set to `TRUE`:

- The window group is disabled and the open window will be not closed or disabled. It will be allowed to continue to its end.

- The window group is disabled but those jobs that have the window group as their schedule will not be disabled.

Job Chains

When a chain is disabled, the metadata for the chain is still there, but jobs that point to it will not be able to be run. This allows changes to the chain to be made safely without the risk of having an incompletely specified chain run. If `force` is set to `FALSE`, the chain must not be referenced by any job, otherwise an error will occur. If `force` is set to `TRUE`, those jobs that point to the chain will not be disabled, however, they will fail at runtime. Running jobs that point to this chain are not affected by the `DISABLE` call and are allowed to complete.

Database Destinations

When you disable a database destination:

- The destination is skipped when a multiple destination job runs.
- If all destinations are disabled for a job, the Scheduler generates an error when it attempts to run the job.
- The `REFS_ENABLED` column in `*_SCHEDULER_JOB_DESTS` is set to `FALSE` for all jobs that reference the database destination.

External Destinations

When you disable an external destination:

- Dependent database destinations remain enabled, but the Scheduler generates an error when it attempts to run a job with a database destination that depends on the external destination.
- The `REFS_ENABLED` column in `*_SCHEDULER_JOB_DESTS` is set to `FALSE` for all external jobs that reference the external destination and for all database jobs with a database destination that depends on the external destination.

Groups

If you disable an external destination group or database destination group, the Scheduler generates an error when it attempts to run a job that names the group as its destination.

DROP_AGENT_DESTINATION Procedure

This procedure drops one or more external destinations, also known as agent destinations. It should be used only when the preferred method of dropping an external destination, using the `schagent` utility to unregister a Scheduler agent with a database, is unavailable due to failures.

This procedure can be called only by the `SYS` user or a user with the `MANAGE_SCHEDULER` privilege.

Note:

External destinations are created on a source database only implicitly by registering an agent with the database. There is no user-callable `CREATE_AGENT_DESTINATION` procedure.

Syntax

```
DBMS_SCHEDULER.DROP_AGENT_DESTINATION (
    destination_name      IN VARCHAR2);
```

Parameters

Table 174-43 DROP_AGENT_DESTINATION Procedure Parameters

Parameter	Description
destination_name	A comma-separated list of external destinations to drop. Because user SYS owns all external destinations, do not prefix them with a schema name. The procedure stops processing if it encounters an external destination that does not exist. All external destinations processed before the error are dropped. Cannot be NULL.

Usage Notes

When an external destination is dropped:

- All database destinations that refer to the external destination are disabled and their `agent` attribute is set to NULL.
- Members of external destination groups that refer to the destination are removed from the group.
- All job instances in the `*_SCHEDULER_JOB_DESTS` views that refer to the external destination are also dropped.
- Jobs running against the destination are stopped.

DROP_CHAIN Procedure

This procedure drops an existing chain.

Syntax

```
DBMS_SCHEDULER.DROP_CHAIN (
    chain_name      IN VARCHAR2,
    force           IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-44 DROP_CHAIN Procedure Parameters

Parameter	Description
chain_name	The name of the chain to drop. Can also be a comma-delimited list of chains.
force	If <code>force</code> is set to FALSE, the chain must not be referenced by any job, otherwise an error will occur. If <code>force</code> is set to TRUE, all jobs pointing to the chain are disabled before the chain is dropped. Running jobs that point to this chain are stopped before the chain is dropped.

Usage Notes

Dropping a chain requires alter privileges on the chain (either as the owner, or a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).

All steps associated with the chain are dropped. If no rule set was specified when the chain was created, then the automatically created rule set and evaluation context associated with the chain are also dropped, so the user must have the privileges required to do this. See the DBMS_RULE_ADM.DROP_RULE_SET and DBMS_RULE_ADM.DROP_EVALUATION_CONTEXT procedures for more information.

If `force` is FALSE, no jobs may be using this chain. If `force` is TRUE, any jobs that use this chain are disabled before the chain is dropped (and any of these jobs that are running will be stopped).

DROP_CHAIN_RULE Procedure

This procedure removes a rule from an existing chain. The rule object corresponding to this rule will also be dropped. The chain will not be disabled. If dropping this rule makes the chain invalid, the user should first disable the chain to ensure that it does not run.

Syntax

```
DBMS_SCHEDULER.DROP_CHAIN_RULE (
    chain_name      IN VARCHAR2,
    rule_name       IN VARCHAR2,
    force           IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-45 DROP_CHAIN_RULE Procedure Parameters

Parameter	Description
chain_name	The name of the chain to alter
rule_name	The name of the rule to drop
force	If <code>force</code> is set to TRUE, the drop operation proceeds even if the chain is currently running. The running chain is not stopped or interrupted. If <code>force</code> is set to FALSE and the chain is running, an error is generated.

Usage Notes

Dropping a chain rule requires alter privileges on the chain (either as the owner or as a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).

Dropping a chain rule also drops the underlying rule database object so you must have the privileges to drop this rule object. See the DBMS_RULE_ADM.DROP_RULE procedure for more information.

DROP_CHAIN_STEP Procedure

This procedure drops a chain step. If this chain step is still used in the chain rules, the chain will be disabled.

Syntax

```
DBMS_SCHEDULER.DROP_CHAIN_STEP (  
    chain_name          IN VARCHAR2,  
    step_name           IN VARCHAR2,  
    force               IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-46 DROP_CHAIN_STEP Procedure Parameters

Parameter	Description
chain_name	The name of the chain to alter
step_name	The name of the step being dropped. Can be a comma-separated list.
force	If <i>force</i> is set to TRUE, this succeeds even if this chain is currently running. The running chain will not be stopped or interrupted. If <i>force</i> is set to FALSE and this chain is currently running, an error is thrown.

Usage Notes

Dropping a chain step requires ALTER privileges on the chain (either as the owner or as a user with ALTER privileges on the chain or the CREATE ANY JOB system privilege).

DROP_CREDENTIAL Procedure

This deprecated procedure drops a credential.



Note:

This procedure is deprecated with Oracle Database 12c Release 1 (12.1). While the procedure remains available in this package, for reasons of backward compatibility, Oracle recommends using the alternative enhanced functionality provided in the [DBMS_CREDENTIAL](#) package, specifically the [DROP_CREDENTIAL Procedure](#).

Syntax

```
DBMS_SCHEDULER.DROP_CREDENTIAL (  
    credential_name     IN VARCHAR2,  
    force               IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-47 DROP_CREDENTIAL Procedure Parameters

Parameter	Description
credential_name	The name of the credential being dropped. This can optionally be prefixed with a schema name. This cannot be set to NULL.
force	If set to FALSE, the credential must not be referenced by any job, or an error will occur. If set to TRUE, the credential is dropped whether or not there are jobs referencing it. Jobs that reference the credential will continue to point to a nonexistent credential and throw an error at runtime.

Usage Notes

Only the owner of a credential or a user with the CREATE ANY JOB system privilege may drop the credential.

Running jobs that point to the credential are not affected by this procedure and are allowed to continue.

DROP_DATABASE_DESTINATION Procedure

This procedure drops one or more database destinations.

Syntax

```
DBMS_SCHEDULER.DROP_DATABASE_DESTINATION (
    destination_name      IN VARCHAR2);
```

Parameters

Table 174-48 DROP_DATABASE_DESTINATION Procedure Parameters

Parameter	Description
destination_name	<p>The name of the destination to drop. Can be a comma-separated list of database destinations to drop. Each database destination can optionally be prefixed with a schema name.</p> <p>The procedure stops processing if it encounters a database destination that does not exist. All database destinations processed before the error are dropped.</p> <p>Cannot be NULL.</p>

Usage Notes

Only the owner or a user with the CREATE ANY JOB system privilege may drop the database destination.

When a database destination is dropped:

- All job instances that refer to the destination in the *_SCHEDULER_JOB_DESTS views are also dropped.
- Jobs running against the destination are stopped.

- Members of database destination groups that refer to the destination are removed from the group.

**See Also:**[CREATE_DATABASE_DESTINATION Procedure](#)

DROP_FILE_WATCHER Procedure

This procedure drops one or more file watchers.

Syntax

```
DBMS_SCHEDULER.DROP_FILE_WATCHER (
    file_watcher_name    IN VARCHAR2,
    force                IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-49 DROP_FILE_WATCHER Procedure Parameters

Parameter	Description
file_watcher_name	The file watcher to drop. Can be a comma-separated list of file watchers. Each file watcher name can optionally be prefixed with a schema name. Cannot be NULL.
force	If set to FALSE, the file watcher must not be referenced by any job, or an error occurs. If set to TRUE, the file watcher is dropped whether or not there are jobs referencing it. In this case, jobs that reference the dropped file watcher are disabled.

Usage Notes

Only the owner of a file watcher or a user with the `CREATE ANY JOB` system privilege may drop the file watcher.

Running jobs that point to the file watcher are not affected by this procedure and are allowed to continue.

**See Also:**["CREATE_FILE_WATCHER Procedure"](#)

DROP_GROUP Procedure

This procedure drops one or more groups.

Syntax

```
DBMS_SCHEDULER.DROP_GROUP (  
    group_name      IN VARCHAR2,  
    force           IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-50 DROP_GROUP Procedure Parameters

Parameter	Description
group_name	A group to drop. Can be a comma-separated list of group names. Each group name can optionally be prefixed with a schema name. The procedure stops processing if it encounters a group that does not exist. All groups processed before the error are dropped. Cannot be NULL.
force	If FALSE, the group must not be referenced by any job, otherwise an error occurs. If TRUE, the group is dropped whether or not there are jobs referencing it. In this case, all jobs referencing the group are disabled and all job instances that reference the group are removed from the *_SCHEDULER_JOB_DESTS views.

Usage Notes

Only the owner or a user with the `CREATE ANY JOB` system privilege may drop a group. You must have the `MANAGE SCHEDULER` privilege to drop a group of type `WINDOW`.



See Also:

["CREATE_FILE_WATCHER Procedure"](#)

DROP_INCOMPATIBILITY Procedure

This procedure drops an existing incompatibility definition.

Syntax

```
DBMS_SCHEDULER.DROP_INCOMPATIBILITY (  
    incompatibility_name  IN VARCHAR2);
```

Parameters

Table 174-51 DROP_INCOMPATIBILITY Procedure Parameters

Parameter	Description
incompatibility_name	The name of the incompatibility definition.

Usage Notes



See Also:

Using Incompatibility Definitions in *Oracle Database Administrator's Guide*

DROP_JOB Procedure

This procedure drops one or more jobs or all jobs in one or more job classes. Dropping a job also drops all argument values set for that job.

Syntax

```
DBMS_SCHEDULER.DROP_JOB (  
    job_name          IN VARCHAR2,  
    force             IN BOOLEAN DEFAULT FALSE,  
    defer             IN BOOLEAN DEFAULT FALSE,  
    commit_semantics  IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters

Table 174-52 DROP_JOB Procedure Parameters

Parameter	Description
job_name	The name of a job or job class. Can be a comma-delimited list. For a job class, the SYS schema should be specified. If the name of a job class is specified, the jobs that belong to that job class are dropped, but the job class itself is not dropped.
force	If <code>force</code> is set to <code>TRUE</code> , the Scheduler first attempts to stop the running job instances (by issuing the <code>STOP_JOB</code> call with the <code>force</code> flag set to <code>false</code>), and then drops the jobs.
defer	If <code>defer</code> is set to <code>TRUE</code> , the Scheduler allows the running jobs to complete and then drops the jobs.

Table 174-52 (Cont.) DROP_JOB Procedure Parameters

Parameter	Description
commit_semantics	<p>The commit semantics. The following types are supported:</p> <ul style="list-style-type: none"> • <code>STOP_ON_FIRST_ERROR</code> returns on the first error and previous successful drop operations are committed to disk. This is the default. • <code>TRANSACTIONAL</code> returns on the first error. Everything that happened before that error is rolled back. This type is not supported when <code>force</code> is set to <code>TRUE</code>. • <code>ABSORB_ERRORS</code> tries to absorb any errors and drop the rest of the jobs, and commits all the successful drops. If errors occur, you can query the view <code>SCHEDULER_BATCH_ERRORS</code> for details. <p>Only <code>STOP_ON_FIRST_ERROR</code> is permitted when job classes are included in the <code>job_name</code> list.</p>

Usage Notes

If both `force` and `defer` are set to `FALSE` and a job is running at the time of the call, the attempt to drop that job fails. The entire call to `DROP_JOB` may then fail, depending on the setting of `commit_semantics`.

Setting both `force` and `defer` to `TRUE` results in an error.

Dropping a job requires `ALTER` privileges on the job either as the owner of the job or as a user with the `ALTER` object privilege on the job or the `CREATE ANY JOB` system privilege.

DROP_JOB_CLASS Procedure

This procedure drops a job class. Dropping a job class means that all the metadata about the job class is removed from the database.

Syntax

```
DBMS_SCHEDULER.DROP_JOB_CLASS (
    job_class_name      IN VARCHAR2,
    force               IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-53 DROP_JOB_CLASS Procedure Parameters

Parameter	Description
job_class_name	The name of the job class. Can be a comma-delimited list.
force	<p>If <code>force</code> is set to <code>FALSE</code>, a class being dropped must not be referenced by any jobs, otherwise an error occurs.</p> <p>If <code>force</code> is set to <code>TRUE</code>, jobs belonging to the class are disabled and their class is set to the default class. Only if this is successful is the class dropped. Running jobs that belong to the job class are not affected.</p>

Usage Notes

Dropping a job class requires the `MANAGE SCHEDULER` system privilege.

DROP_PROGRAM Procedure

This procedure drops a program. Any arguments that pertain to the program are also dropped when the program is dropped.

Syntax

```
DBMS_SCHEDULER.DROP_PROGRAM (  
    program_name          IN VARCHAR2,  
    force                 IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-54 DROP_PROGRAM Procedure Parameters

Parameter	Description
program_name	The name of the program to be dropped. Can be a comma-delimited list.
force	<p>If <i>force</i> is set to FALSE, the program must not be referenced by any job, otherwise an error occurs.</p> <p>If <i>force</i> is set to TRUE, all jobs referencing the program are disabled before the program is dropped.</p> <p>Running jobs that point to the program are not affected by the DROP_PROGRAM call and are allowed to continue.</p>

Usage Notes

Dropping a program requires that you be the owner of the program or have **ALTER** privileges on that program. You can also drop a program if you have the **CREATE ANY JOB** privilege.

DROP_PROGRAM_ARGUMENT Procedure

This procedure drops a program argument. An argument can be specified by either name (if one has been given) or position.

The procedure is overloaded.

Syntax

Drops a program argument by position:

```
DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (  
    program_name          IN VARCHAR2,  
    argument_position     IN PLS_INTEGER);
```

Drops a program argument by name:

```
DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (  
    program_name          IN VARCHAR2,  
    argument_name         IN VARCHAR2);
```

Parameters

Table 174-55 DROP_PROGRAM_ARGUMENT Procedure Parameters

Parameter	Description
program_name	The name of the program to be altered. A program with this name must exist.
argument_name	The name of the argument being dropped
argument_position	The position of the argument to be dropped

Usage Notes

Dropping a program argument requires that you be the owner of the program or have `ALTER` privileges on that program. You can also drop a program argument if you have the `CREATE ANY JOB` privilege.

DROP_RESOURCE Procedure

This procedure drops a resource.

Syntax

```
DBMS_SCHEDULER.DROP_RESOURCE (  
    resource_name  IN VARCHAR2,  
    force          IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-56 DROP_RESOURCE Procedure Parameters

Parameter	Description
resource_name	The name of the resource to be dropped. Can be a comma-delimited list.
force	If <code>force</code> is set to <code>FALSE</code> , the resource must not have any existing constraints, otherwise an error occurs. If <code>force</code> is set to <code>TRUE</code> , the resource will be dropped and any constraints defined on this resource will also be dropped.

Usage Notes

Only the owner or a user with the `CREATE ANY JOB` system privilege may drop the resource.



See Also:

Creating or Dropping a Resource in *Oracle Database Administrator's Guide*

DROP_SCHEDULE Procedure

This procedure drops a schedule.

Syntax

```
DBMS_SCHEDULER.DROP_SCHEDULE (  
    schedule_name    IN VARCHAR2,  
    force             IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-57 DROP_SCHEDULE Procedure Parameters

Parameter	Description
schedule_name	The name of the schedule. Can be a comma-delimited list.
force	If <code>force</code> is set to <code>FALSE</code> , the schedule must not be referenced by any job or window, otherwise an error will occur. If <code>force</code> is set to <code>TRUE</code> , any jobs or windows that use this schedule are disabled before the schedule is dropped Running jobs and open windows that point to the schedule are not affected.

Usage Notes

You must be the owner of the schedule being dropped or have `ALTER` privileges for the schedule or the `CREATE ANY JOB` privilege.

DROP_WINDOW Procedure

This procedure drops a window. All metadata about the window is removed from the database. The window is removed from any groups that reference it.

Syntax

```
DBMS_SCHEDULER.DROP_WINDOW (  
    window_name      IN VARCHAR2,  
    force             IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-58 DROP_WINDOW Procedure Parameters

Parameter	Description
window_name	The name of the window. Can be a comma-delimited list.

Table 174-58 (Cont.) DROP_WINDOW Procedure Parameters

Parameter	Description
<code>force</code>	<p>If <code>force</code> is set to <code>FALSE</code>, the window must be not be open or referenced by any job, otherwise an error occurs.</p> <p>If <code>force</code> is set to <code>TRUE</code>, the window is dropped and those jobs that have the window as their schedule are disabled. However, jobs that have a window group, of which the dropped window is a member, as their schedule, are not disabled. If the window is open then, the Scheduler attempts to first close the window and then drop it. When the window is closed, normal close window rules apply.</p> <p>Running jobs that have the window as their schedule is allowed to continue, unless the <code>stop_on_window_close</code> flag is set to <code>TRUE</code> for the job. If this is the case, the job is stopped when the window is dropped.</p>

Usage Notes

Dropping a window requires the `MANAGE SCHEDULER` privilege.

DUMP_IN_MEMORY_TRACE Procedure

This procedure dumps the scheduler in-memory trace buffer of the specified process state object address into the current trace file of the requester process.

Syntax

```
DBMS_SCHEDULER.DUMP_IN_MEMORY_TRACE (
    PROCESS_ADDRESS IN RAW);
```

Parameters

Table 174-59 DUMP_IN_MEMORY_TRACE Procedure Parameters

Parameter	Description
<code>PROCESS_ADDRESS</code>	State object address of the process being dumped.

Usage Notes

Either connecting as a `SYS` user or having the `DBA` role is required to execute `dump_in_memory_trace`.

ENABLE Procedure

This procedure enables a program, job, chain, window, database destination, external destination, file watcher, or group.

When an object is enabled, its `enabled` attribute is set to `TRUE`. By default, jobs, chains, and programs are created disabled and database destinations, external destinations, file watchers, windows, and groups are created enabled.

If a job was disabled and you enable it, the Scheduler begins to automatically run the job according to its schedule. Enabling a disabled job also resets the job `RUN_COUNT`, `FAILURE_COUNT` and `RETRY_COUNT` columns in the `*_SCHEDULER_JOBS` data dictionary views.

Validity checks are performed before enabling an object. If the check fails, the object is not enabled, and an appropriate error is returned. This procedure does not return an error if the object was already enabled.

Syntax

```
DBMS_SCHEDULER.ENABLE (  
    name                IN VARCHAR2,  
    commit_semantics    IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters

Table 174-60 ENABLE Procedure Parameters

Parameter	Description
name	<p>The name of the Scheduler object being enabled. Can be a comma-delimited list of names.</p> <p>If a job class name is specified, then all the jobs in the job class are enabled.</p> <p>If a group name is specified, then the group is enabled, but the enabled state of the group members is unaffected.</p>
commit_semantics	<p>The commit semantics. The following types are supported:</p> <ul style="list-style-type: none">• <code>STOP_ON_FIRST_ERROR</code> - The procedure returns on the first error and previous successful enable operations are committed to disk. This is the default.• <code>TRANSACTIONAL</code> - The procedure returns on the first error and everything that happened before that error is rolled back. This type is only supported when enabling a job or a list of jobs.• <code>ABSORB_ERRORS</code> - The procedure tries to absorb any errors and enable the rest of the jobs. It commits all the enable operations that were successful. If errors occur, you can query the view <code>SCHEDULER_BATCH_ERRORS</code> for details. This type is only supported when enabling a job or a list of jobs.

Usage Notes

Window names must be preceded by `SYS`.

To run `ENABLE` for a window or group of type `WINDOW`, you must have the `MANAGE SCHEDULER` privilege. For a job of type `EXECUTABLE` (or for a job that points to a program of type `EXECUTABLE`), the job owner must have the `CREATE EXTERNAL JOB` system privilege before the job can be enabled or run.

To enable a file watcher, the file watcher owner must have the `EXECUTE` privilege on the designated credential.

You can use `ENABLE` with any schema except the `SYS` schema.

END_DETACHED_JOB_RUN Procedure

This procedure ends a detached job run. A detached job points to a detached program, which is a program with the `detached` attribute set to `TRUE`.

A detached job run does not end until this procedure or the [STOP_JOB Procedure](#) is called.

Syntax

```
DBMS_SCHEDULER.END_DETACHED_JOB_RUN (  
    job_name          IN VARCHAR2,  
    error_number      IN PLS_INTEGER DEFAULT 0,  
    additional_info    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-61 END_DETACHED_JOB_RUN Procedure Parameters

Parameter	Description
job_name	The name of the job to end. Must be a detached job that is running.
error_number	If zero, then the job run is logged as succeeded. If -1013, then the job run is logged as stopped. If any other number, then the job run is logged as failed with that error number.
additional_info	This text is stored in the additional_info column of the *_scheduler_job_run_details views for this job run.

Usage Notes

This procedure requires that you either own the job or have `ALTER` privileges on it. You can also end any detached job run if you have the `CREATE ANY JOB` privilege.



See Also:

Oracle Database Administrator's Guide for information about detached jobs.

EVALUATE_CALENDAR_STRING Procedure

You can define repeat intervals of jobs, windows or schedules using the Scheduler calendaring syntax. This procedure evaluates the calendar expression and tells you the next execution date and time of a job or window. This is very useful for testing the correct definition of the calendar string without actually scheduling the job or window.

This procedure can also get multiple steps of the repeat interval by passing the `next_run_date` returned by one invocation as the `return_date_after` argument of the next invocation.

See the calendaring syntax described in "[Operational Notes](#)".

Syntax

```
DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING (  
    calendar_string    IN VARCHAR2,  
    start_date         IN TIMESTAMP WITH TIME ZONE,  
    return_date_after  IN TIMESTAMP WITH TIME ZONE,  
    next_run_date      OUT TIMESTAMP WITH TIME ZONE);
```

Parameters

Table 174-62 EVALUATE_CALENDAR_STRING Procedure Parameters

Parameter	Description
calendar_string	The calendar expression to be evaluated. The string must be in the calendaring syntax described in "Operational Notes" .
start_date	The date and time after which the repeat interval becomes valid. It can also be used to fill in specific items that are missing from the calendar string. Can optionally be NULL.
return_date_after	The return_date_after argument helps the Scheduler determine which one of all possible matches (all valid execution dates) to return from those determined by the start_date and the calendar string. When a NULL value is passed for this argument, the Scheduler automatically fills in systimestamp as its value.
next_run_date	The first timestamp that matches the calendar string and start date that occur after the value passed in for the return_date_after argument.

Examples

The following code fragment can be used to determine the next five dates a job will run given a specific calendar string.

```
SET SERVEROUTPUT ON;
ALTER SESSION set NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
Session altered.

DECLARE
start_date          TIMESTAMP;
return_date_after   TIMESTAMP;
next_run_date       TIMESTAMP;
BEGIN
start_date :=
to_timestamp_tz('01-JAN-2003 10:00:00','DD-MON-YYYY HH24:MI:SS');
return_date_after := start_date;
FOR i IN 1..5 LOOP
    DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING(
        'FREQ=DAILY;BYHOUR=9;BYMINUTE=30;BYDAY=MON,TUE,WED,THU,FRI',
        start_date, return_date_after, next_run_date);
    DBMS_OUTPUT.PUT_LINE('next_run_date: ' || next_run_date);
    return_date_after := next_run_date;
END LOOP;
END;
/

next_run_date: 02-JAN-03 09.30.00.000000 AM
next_run_date: 03-JAN-03 09.30.00.000000 AM
next_run_date: 06-JAN-03 09.30.00.000000 AM
next_run_date: 07-JAN-03 09.30.00.000000 AM
next_run_date: 08-JAN-03 09.30.00.000000 AM
```

PL/SQL procedure successfully completed.

Usage Notes

No specific Scheduler privileges are required.

EVALUATE_RUNNING_CHAIN Procedure

This procedure forces reevaluation of the rules of a running chain to trigger any rules for which the conditions have been satisfied. The job passed as an argument must point to a chain and must be running. If the job is not running, an error is thrown. (`RUN_JOB` can be used to start the job.)

If any of the steps of the chain are themselves running chains, another `EVALUATE_RUNNING_CHAIN` is performed on each of the nested running chains.

Syntax

```
DBMS_SCHEDULER.EVALUATE_RUNNING_CHAIN (  
    job_name          IN VARCHAR2);
```

Parameters

Table 174-63 EVALUATE_RUNNING_CHAIN Procedure Parameter

Parameter	Description
job_name	The name of the running job (pointing to a chain) to reevaluate the rules for

Usage Notes

Running `EVALUATE_RUNNING_CHAIN` on a job requires alter privileges on the job (either as the owner, or as a user with `ALTER` privileges on the job or the `CREATE ANY JOB` system privilege).



Note:

The Scheduler automatically evaluates a chain:

- At the start of the chain job
- When a chain step completes
- When an event occurs that is associated with an event step in the chain

For most chains, this is sufficient. `EVALUATE_RUNNING_CHAIN` should be used only under the following circumstances:

- After manual intervention of a running chain with the `ALTER_RUNNING_CHAIN` procedure
- When chain rules use SQL syntax and the rule conditions contain elements that are not under the control of the Scheduler.

In these cases, `EVALUATE_RUNNING_CHAIN` may not be needed if you set the `evaluation_interval` attribute when you created the chain.

GENERATE_JOB_NAME Function

This function returns a unique name for a job.

The name will be of the form {prefix}N where N is a number from a sequence. If no prefix is specified, the generated name will, by default, be JOB\$_1, JOB\$_2, JOB\$_3, and so on. If 'SCOTT' is specified as the prefix, the name will be SCOTT1, SCOTT2, and so on.

Syntax

```
DBMS_SCHEDULER.GENERATE_JOB_NAME (  
    prefix          IN VARCHAR2 DEFAULT 'JOB$_') RETURN VARCHAR2;
```

Parameters

Table 174-64 GENERATE_JOB_NAME Function Parameter

Parameter	Description
prefix	The prefix to use when generating the job name

Usage Notes

If the prefix is explicitly set to `NULL`, the name is just the sequence number. In order to successfully use such numeric names, they must be surrounded by double quotes throughout the `DBMS_SCHEDULER` calls. A prefix cannot be longer than 18 characters and cannot end with a digit.

Note that, even though the `GENERATE_JOB_NAME` function never returns the same job name twice, there is a small chance that the returned name matches an already existing database object.

No specific Scheduler privileges are required to use this function.

GET_AGENT_INFO Function

This function can return job information specific to an agent, such as how many are running and so on, depending on the attribute selected.

Syntax

```
DBMS_SCHEDULER.GET_AGENT_INFO (  
    agent_name      IN VARCHAR2,  
    attribute        IN VARCHAR2) RETURN VARCHAR2;
```

Parameters

Table 174-65 GET_AGENT_INFO Function Parameter

Parameter	Description
agent_name	The name of an external destination where the agent is running

Table 174-65 (Cont.) GET_AGENT_INFO Function Parameter

Parameter	Description
attribute	<p>Possible Attributes values</p> <ul style="list-style-type: none"> • VERSION: Returns the agent version number. Requires the <code>CREATE JOB</code> system privilege. • UPTIME: Returns the time the agent has been up and running. Requires the <code>CREATE JOB</code> system privilege. • NUMBER_OF_RUNNING_JOBS: Returns the number of jobs that the agent is currently running. Requires the <code>CREATE JOB</code> system privilege. • TOTAL_JOBS_RUN: Returns the number of jobs run by the agent since it was started. Requires the <code>CREATE JOB</code> system privilege. • RUNNING_JOBS: Returns a comma-separated list of the names of the jobs running currently. Requires the <code>MANAGE SCHEDULER</code> system privilege. • ALL: Returns all the information the previous options return. It requires the <code>MANAGE SCHEDULER</code> system privilege.

Usage Notes

This function returns the same information as the `schagent` utility status option. See *Oracle Database Administrator's Guide*.

GET_AGENT_VERSION Function

This function returns the version string of a Scheduler agent that is registered with the database and is currently running. `GET_AGENT_VERSION` throws an error if the agent is not registered with the database or if the agent is not currently running.

Syntax

```
DBMS_SCHEDULER.GET_AGENT_VERSION (
    agent_host      IN VARCHAR2) RETURN VARCHAR2;
```

Parameters

Table 174-66 GET_AGENT_VERSION Function Parameter

Parameter	Description
agent_host	Either the hostname and port on which the agent is running in the form <code>hostname:port</code> or the name of the agent as shown in the <code>destination_name</code> column of the <code>ALL_SCHEDULER_EXTERNAL_DESTS</code> view which lists all Scheduler agents registered with the database.

Usage Notes

This function requires the `CREATE EXTERNAL JOB` system privilege.

GET_ATTRIBUTE Procedure

This procedure retrieves the value of an attribute of a Scheduler object. It is overloaded to retrieve values of various types.

Syntax

```
DBMS_SCHEDULER.GET_ATTRIBUTE (
    name          IN VARCHAR2,
    attribute     IN VARCHAR2,
    value         OUT {VARCHAR2|PLS_INTEGER|BOOLEAN|DATE|TIMESTAMP|
                     TIMESTAMP WITH TIME ZONE|TIMESTAMP WITH LOCAL TIME ZONE|
                     INTERVAL DAY TO SECOND});
```

```
DBMS_SCHEDULER.GET_ATTRIBUTE (
    name          IN VARCHAR2,
    attribute     IN VARCHAR2,
    value         OUT VARCHAR2,
    value2        OUT VARCHAR2);
```

Parameters

Table 174-67 GET_ATTRIBUTE Procedure Parameters

Parameter	Description
name	The name of the object
attribute	The attribute being retrieved. See the SET_ATTRIBUTE Procedure for tables of attribute values.
value	The existing value of the attribute
value2	The value2 argument is for an optional second value. Most attributes have only one value associated with them, but some can have two.

Usage Notes

To run `GET_ATTRIBUTE` for a job class, you must have the `MANAGE SCHEDULER` privilege or have `EXECUTE` privileges on the class. For a schedule, window, or group, no privileges are necessary. Otherwise, you must be the owner of the object or have `ALTER` or `EXECUTE` privileges on that object or have the `CREATE ANY JOB` privilege.

See the [SET_ATTRIBUTE Procedure](#) for tables of attribute values that you can retrieve for the various Scheduler object types.

GET_FILE Procedure

This procedure retrieves a file from the operating system file system of a specified host. The file is copied to a destination, or its contents are returned in a procedure output parameter.

You can also use this procedure to retrieve the standard output or error text for a run of an external job that has an associated credential.

This procedure differs from the equivalent `UTL_FILE` procedure in that it uses a credential and can retrieve files from remote hosts that have only a Scheduler agent (and not an Oracle database) installed.

Syntax

```
DBMS_SCHEDULER.GET_FILE (
    source_file          IN VARCHAR2,
    source_host          IN VARCHAR2,
    credential_name      IN VARCHAR2,
    file_contents        IN OUT NOCOPY {BLOB|CLOB});
```

```
DBMS_SCHEDULER.GET_FILE (
    source_file          IN VARCHAR2,
    source_host          IN VARCHAR2,
    credential_name      IN VARCHAR2,
    destination_file_name IN VARCHAR2,
    destination_directory_object IN VARCHAR2,
    destination_permissions IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-68 GET_FILE Procedure Parameters

Parameter	Description
source_file	<p>Fully qualified path name of the file to retrieve from the operating system. The file name is case-sensitive and is not converted to uppercase. If the file name starts with a question mark ('?'), the question mark is replaced by the path to the Oracle home if getting a file from the local host, or to the Scheduler agent home if getting a file from a remote host.</p> <p>If the format of this parameter is <i>external_log_id_stdout</i>, then the stdout from the designated external job run is returned.</p> <p>If the format of this parameter is <i>external_log_id_stderr</i>, the error text from the designated external job run is returned.</p> <p>You obtain the value of <i>external_log_id</i> from the <i>ADDITIONAL_INFO</i> column of the <i>*_SCHEDULER_JOB_RUN_DETAILS</i> views. This column contains a set of name/value pairs in an indeterminate order, so you must parse this column for the <i>external_log_id</i> name/value pair, and then append either <i>"_stdout"</i> or <i>"_stderr"</i> to its value.</p> <p>The external job must have an associated credential. The <i>credential_name</i> parameter of <i>GET_FILE</i> must name the same credential that is used by the job, and the <i>source_host</i> parameter must be the same as the <i>destination</i> attribute of the job.</p>
source_host	<p>If the file is to be retrieved from a remote host, then this parameter must be a valid an external destination name. (An external destination is created when you register a remote Scheduler agent with the database. You can view external destination names in the views <i>*_SCHEDULER_EXTERNAL_DESTS</i>.)</p> <p>If <i>source_host</i> is <i>NULL</i> or set to <i>'localhost'</i>, then the file is retrieved from the file system of the local host. To determine the port number of a Scheduler agent, view the <i>schagent.conf</i> file, which is located in the Scheduler agent home directory on the remote host.</p>
credential_name	The name of the credential to use for accessing the file system.
file_contents	The variable into which the file contents is read.
destination_file_name	The file to which the file contents is written.
destination_directory_object	The directory object that specifies the path to the destination file, when <i>destination_file_name</i> is used. The caller must have the necessary privileges on the directory object.

Table 174-68 (Cont.) GET_FILE Procedure Parameters

Parameter	Description
destination_permissions	Reserved for future use

Usage Notes

The caller must have the `CREATE EXTERNAL JOB` system privilege and have `EXECUTE` privileges on the credential.

GET_SCHEDULER_ATTRIBUTE Procedure

This procedure retrieves the value of a Scheduler attribute.

Syntax

```
DBMS_SCHEDULER.GET_SCHEDULER_ATTRIBUTE (  
    attribute    IN VARCHAR2,  
    value        OUT VARCHAR2);
```

Parameters**Table 174-69 GET_SCHEDULER_ATTRIBUTE Procedure Parameters**

Parameter	Description
attribute	The name of the attribute
value	The existing value of the attribute

Usage Notes

To run `GET_SCHEDULER_ATTRIBUTE`, you must have the `MANAGE SCHEDULER` privilege.

[Table 174-70](#) lists the Scheduler attributes that you can retrieve. For more detail on these attributes, see [Table 174-102](#) and the section "Configuring the Scheduler" in *Oracle Database Administrator's Guide*.

Table 174-70 Scheduler Attributes Retrievable with GET_SCHEDULER_ATTRIBUTE

Scheduler Attribute	Description
current_open_window	Name of the currently open window
default_timezone	Default time zone used by the Scheduler for repeat intervals and windows
email_sender	The default e-mail address of the sender for job state e-mail notifications
email_server	The SMTP server address that the Scheduler uses to send e-mail notifications for job state events. E-mail notifications cannot be sent if this attribute is <code>NULL</code> .
event_expiry_time	Time in seconds before an event generated by the Scheduler and enqueued onto the Scheduler event queue expires. May be <code>NULL</code> .

**Table 174-70 (Cont.) Scheduler Attributes Retrievable with
GET_SCHEDULER_ATTRIBUTE**

Scheduler Attribute	Description
log_history	Retention period in days for job and window logs. The range of valid values is 0 through 1000000.
max_job_slave_processes	This Scheduler attribute is not used.

OPEN_WINDOW Procedure

This procedure manually opens a window, unrelated to its schedule.

The window opens and the resource plan associated with it takes effect immediately, for the duration specified or for the normal duration of the window, if no duration is given. Only an enabled window can be manually opened.

Syntax

```
DBMS_SCHEDULER.OPEN_WINDOW (  
    window_name          IN VARCHAR2,  
    duration              IN INTERVAL DAY TO SECOND,  
    force                 IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 174-71 OPEN_WINDOW Procedure Parameters

Parameter	Description
window_name	The name of the window
duration	The duration of the window. It is of type interval day to second. If it is <code>NULL</code> , then the window opens for the regular duration as specified in the window metadata.
force	<p>If <code>force</code> is set to <code>FALSE</code>, then opening an already open window generates an error.</p> <p>If <code>force</code> is set to <code>TRUE</code>:</p> <p>You can open a window that is already open. The window stays open for the duration specified in the call, from the time the <code>OPEN_WINDOW</code> command was issued.</p> <p>For example: <code>window1</code> was created with a duration of four hours. It has now been open for two hours. If, at this point, you reopen <code>window1</code> using the <code>OPEN_WINDOW</code> call and do not specify a duration, then <code>window1</code> stays open for four hours because it was created with that duration. If you specified a duration of 30 minutes, the window will close in 30 minutes.</p> <p>The Scheduler automatically closes any window that is open at that time, even if it has a higher priority. For the duration of this manually opened window, the Scheduler does not open any other scheduled windows even if they have a higher priority.</p>

Usage Notes

Opening a window manually has no impact on regular scheduled runs of the window. The next open time of the window is not updated and is determined by the regular scheduled opening.

When a window that was manually opened closes, the rules about overlapping windows are applied to determine which other window should be opened at that time if any at all.

If there are jobs running when the window opens, the resources allocated to them might change if there is a switch in resource plan.

If a window fails to switch resource plans because the designated resource plan no longer exists or because resource plan switching by windows is disabled (for example, by using the `ALTER SYSTEM` statement with the `force` option), the failure to switch resource plans is recorded in the window log.

Opening a window requires the `MANAGE SCHEDULER` privilege.

PURGE_LOG Procedure

The `PURGE_LOG` procedure purges rows from the job and window log that were not purged automatically by the scheduler.

By default, the Scheduler automatically purges all rows in the job log and window log that are older than 30 days. The `PURGE_LOG` procedure can be used to purge additional rows from the job and window log.

Rows in the job log table pertaining to the steps of a chain are purged only when the entry for the main chain job is purged (either manually or automatically).

Syntax

```
DBMS_SCHEDULER.PURGE_LOG (
    log_history          IN PLS_INTEGER  DEFAULT 0,
    which_log           IN VARCHAR2     DEFAULT 'JOB_AND_WINDOW_LOG',
    job_name            IN VARCHAR2     DEFAULT NULL);
```

Parameters

Table 174-72 PURGE_LOG Procedure Parameters

Parameter	Description
<code>log_history</code>	This specifies how much history (in days) to keep. The valid range is 0 - 1000000. If set to 0, no history is kept.
<code>which_log</code>	This specifies the log type. Valid values are: <code>job_log</code> , <code>window_log</code> , and <code>job_and_window_log</code> .
<code>job_name</code>	This specifies which job-specific entries must be purged from the job log. This can be a comma-delimited list of job names and job classes. Whenever <code>job_name</code> has a value other than <code>NULL</code> , the <code>which_log</code> argument implicitly includes the job log.

Usage Notes

This procedure requires the `MANAGE SCHEDULER` privilege.

Examples

The following completely purges all rows from both the job log and the window log:

```
DBMS_SCHEDULER.PURGE_LOG();
```

The following purges all rows from the window log that are older than 5 days:

```
DBMS_SCHEDULER.PURGE_LOG(5, 'window_log');
```

The following purges all rows from the window log that are older than 1 day and all rows from the job log that are related to jobs in `jobclass1` and older than 1 day:

```
DBMS_SCHEDULER.PURGE_LOG(1, 'job_and_window_log', 'sys.jobclass1');
```

PUT_FILE Procedure

This procedure saves a file to the operating system file system of a specified remote host or of the local computer.

It differs from the equivalent `UTL_FILE` procedure in that it uses a credential and can save files to a remote host that has only a Scheduler agent (and not an Oracle Database) installed.

Syntax

```
DBMS_SCHEDULER.PUT_FILE (
    destination_file          IN VARCHAR2,
    destination_host          IN VARCHAR2,
    credential_name           IN VARCHAR2,
    file_contents              IN {BLOB|CLOB},
    destination_permissions   IN VARCHAR2 DEFAULT NULL);

DBMS_SCHEDULER.PUT_FILE (
    destination_file          IN VARCHAR2,
    destination_host          IN VARCHAR2,
    credential_name           IN VARCHAR2,
    source_file_name           IN VARCHAR2,
    source_directory_object    IN VARCHAR2,
    destination_permissions   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-73 PUT_FILE Procedure Parameters

Parameter	Description
<code>destination_file</code>	Fully qualified path name of the file to save to the operating system file system. The file name is case-sensitive. If the file name starts with a question mark ('?'), the question mark is replaced by the path to the Oracle home if saving to the local host, or to the Scheduler agent home if saving to a remote host.
<code>destination_host</code>	If <code>NULL</code> or set to 'localhost', the file is saved to the file system of the local computer. To save to a remote host, this parameter must be a valid external destination name. (An external destination is created when you register a remote Scheduler agent with the database. You can view external destination names in the views <code>*_SCHEDULER_EXTERNAL_DESTS</code> .)
<code>credential_name</code>	The name of the credential to use for accessing the destination file system.
<code>file_contents</code>	The variable from which the file contents is read.
<code>source_file_name</code>	The file from which the file contents is written
<code>source_directory_object</code>	The directory object that specifies the path to the source file, when <code>source_file_name</code> is used. The caller must have the necessary privileges on the directory object.
<code>destination_permissions</code>	Reserved for future use

Usage Notes

The caller must have the `CREATE EXTERNAL JOB` system privilege and have `EXECUTE` privileges on the credential.

REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure

This procedure unsubscribes a user from the Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE`.

Syntax

```
DBMS_SCHEDULER.REMOVE_EVENT_QUEUE_SUBSCRIBER (  
    subscriber_name      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-74 REMOVE_EVENT_QUEUE_SUBSCRIBER Procedure Parameters

Parameter	Description
<code>subscriber_name</code>	Name of the Oracle Advanced Queuing (AQ) agent to remove the subscription from. If <code>NULL</code> , the user name of the calling user is used.

Usage Notes

After the agent is unsubscribed, it is deleted. If the agent does not exist or is not currently subscribed to the Scheduler event queue, an error is raised.

REMOVE_FROM_INCOMPATIBILITY Procedure

This procedure removes jobs or programs from an existing incompatibility definition.

Syntax

```
DBMS_SCHEDULER.REMOVE_FROM_INCOMPATIBILITY (  
    incompatibility_name  IN VARCHAR2,  
    object_name           IN VARCHAR2);
```

Parameters

Table 174-75 REMOVE_FROM_INCOMPATIBILITY Procedure Parameters

Parameter	Description
<code>incompatibility_name</code>	The name of the incompatibility definition.
<code>object_name</code>	One or more (comma-separated) programs or jobs

Usage Notes

This procedure does not raise an error if any specified objects do not already exist in the incompatibility definition.

**See Also:**

Using Incompatibility Definitions in *Oracle Database Administrator's Guide*

REMOVE_GROUP_MEMBER Procedure

This procedure removes one or more members from an existing group.

Syntax

```
DBMS_SCHEDULER.REMOVE_GROUP_MEMBER (  
    group_name          IN VARCHAR2,  
    member              IN VARCHAR2);
```

Parameters

Table 174-76 REMOVE_GROUP_MEMBER Procedure Parameters

Parameter	Description
group_name	The name of the group.
member_name	<p>The name of the member to remove from group. Comma-separated list of members to remove. An error is returned if any of the members is not part of the group.</p> <p>A group of the same type can be named as a member. The Scheduler immediately expands the included group name into its list of members.</p> <p>If the member is a destination, any job instances that run on this destination are removed from the *_SCHEDULER_JOB_DESTS views.</p>

Usage Notes

The following users may remove members from a group:

- The group owner
- A user that has been granted the ALTER object privilege on the group
- A user with the CREATE ANY JOB system privilege

You must have the MANAGE SCHEDULER privilege to remove a member from a group of type WINDOW.

**See Also:**

["CREATE_GROUP Procedure"](#)

REMOVE_JOB_EMAIL_NOTIFICATION Procedure

This procedure removes e-mail notifications for a job. You can remove all e-mail notifications or remove notifications only for specified recipients or specified events.

Syntax

```
DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION (
    job_name          IN VARCHAR2,
    recipients        IN VARCHAR2 DEFAULT NULL,
    events            IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-77 ADD_JOB_EMAIL_NOTIFICATION Procedure Parameters

Parameter	Description
job_name	Name of the job to remove e-mail notifications for. Cannot be NULL.
recipients	E-mail address to remove e-mail notification for. Comma-separated list of e-mail addresses.
events	Job state event to remove e-mail notification for. Comma-separate list of job state events.

Usage Notes

When you specify multiple recipients and multiple events, the notification for each specified event is removed for each specified recipient. The procedure ignores any recipients or events that are specified but that were not previously added.

If `recipients` is NULL, e-mail notifications for the specified events are removed for all existing recipients. If `events` is NULL, notifications for all events are removed for the specified recipients. If both `recipients` and `events` are NULL, all e-mail notifications are removed for the job.

For example, if `recipients` is 'jsmith@example.com,rjones@example.com' and `events` is 'JOB_FAILED,JOB_BROKEN', then notifications for both the `JOB_FAILED` and `JOB_BROKEN` events are removed for both jsmith and rjones. If `recipients` is NULL, then notifications for both the `JOB_FAILED` and `JOB_BROKEN` events are removed for jsmith, rjones, and any other previously defined recipients for these events.

To call this procedure, you must be the job owner or a user with the `CREATE ANY JOB` system privilege or `ALTER` object privilege on the job.



See Also:

["ADD_JOB_EMAIL_NOTIFICATION Procedure"](#)

RESET_JOB_ARGUMENT_VALUE Procedure

This procedure resets (clears) the value previously set to an argument for a job.

`RESET_JOB_ARGUMENT_VALUE` is overloaded.

Syntax

Clears a previously set job argument value by argument position:

```
DBMS_SCHEDULER.RESET_JOB_ARGUMENT_VALUE (
    job_name          IN VARCHAR2,
    argument_position IN PLS_INTEGER);
```

Clears a previously set job argument value by argument name:

```
DBMS_SCHEDULER.RESET_JOB_ARGUMENT_VALUE (
    job_name          IN VARCHAR2,
    argument_name     IN VARCHAR2);
```

Parameters

Table 174-78 RESET_JOB_ARGUMENT_VALUE Procedure Parameters

Parameter	Description
job_name	The name of the job being altered
argument_position	The position of the program argument being reset
argument_name	The name of the program argument being reset

Usage Notes

If the corresponding program argument has no default value, the job is disabled. Resetting a program argument of a job belonging to another user requires **ALTER** privileges on that job. Arguments can be specified by position or by name.

RESET_JOB_ARGUMENT_VALUE requires that you be the owner of the job or have **ALTER** privileges on that job. You can also reset a job argument value if you have the **CREATE ANY JOB** privilege.

RESET_JOB_ARGUMENT_VALUE only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.

RUN_CHAIN Procedure

This procedure immediately runs a chain or part of a chain by creating a run-once job with the job name given.

If no **job_name** is given, one is generated of the form **RUN_CHAIN\$_chainnameN**, where *chainname* is the first 8 characters of the chain name and *N* is an integer.

If a list of start steps is given, only those steps are started when the chain begins running. Steps not in the list that would normally have started are skipped and paused (so that they or the steps after them do not run).

If **start_steps** is **NULL**, then the chain starts normally—that is, it performs an initial evaluation to see which steps to start running).

If a list of initial step states is given, the newly created chain job sets every listed step to the state specified for that step before evaluating the chain rules to see which steps to start. (Steps in the list are not started.)

Syntax

Runs a chain, with a list of start steps.

```
DBMS_SCHEDULER.RUN_CHAIN (
    chain_name          IN VARCHAR2,
    start_steps         IN VARCHAR2,
    job_name            IN VARCHAR2 DEFAULT NULL);
```

Runs a chain, with a list of initial step states.

```
DBMS_SCHEDULER.RUN_CHAIN (
    chain_name          IN VARCHAR2,
    step_state_list     IN SYS.SCHEDULER$_STEP_TYPE_LIST,
    job_name            IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 174-79 RUN_CHAIN Procedure Parameters

Parameter	Description
chain_name	The name of the chain to run
job_name	The name of the job to create to run the chain
start_steps	Comma-separated list of the steps to start when the chain starts running
step_state_list	List of chain steps with an initial state (SUCCEEDED or FAILED) to set for each. Set the attributes of sys.scheduler\$_step_type as follows: step_name The name of the step step_type 'SUCCEEDED' or 'FAILED' error_number where error_number is a positive or negative integer.

Usage Notes

Running a chain requires `CREATE JOB` if the job is being created in the user's schema, or `CREATE ANY JOB` otherwise. In addition, the owner of the job being created needs execute privileges on the chain (as the owner of the chain, or as a user with the `EXECUTE` privilege on the chain or the `EXECUTE ANY PROGRAM` system privilege).

Examples

The following example illustrates how to start a chain in the middle by providing the initial state of some chain steps.

```
declare
    initial_step_states sys.scheduler$_step_type_list;
begin
    initial_step_states := sys.scheduler$_step_type_list(
        sys.scheduler$_step_type('step1', 'SUCCEEDED'),
        sys.scheduler$_step_type('step2', 'FAILED 27486'),
        sys.scheduler$_step_type('step3', 'SUCCEEDED'),
        sys.scheduler$_step_type('step5', 'SUCCEEDED'));
    dbms_scheduler.run_chain('my_chain', initial_step_states);
end;
/
```

RUN_JOB Procedure

This procedure runs a job immediately.

If a job is enabled, the Scheduler runs it automatically. It is not necessary to call `RUN_JOB` to run a job according to its schedule. Use `RUN_JOB` to run a job outside of its normal schedule.

Syntax

```
DBMS_SCHEDULER.RUN_JOB (
    job_name          IN VARCHAR2,
    use_current_session IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 174-80 RUN_JOB Procedure Parameters

Parameter	Description
<code>job_name</code>	<p>A job name or a comma-separate list of entries, where each is the name of an existing job, optionally preceded by a schema name and dot separator.</p> <p>If you specify a multiple-destination job, the job runs on all destinations. In this case, the <code>use_current_session</code> argument must be <code>FALSE</code>.</p>
<code>use_current_session</code>	<p>This specifies whether or not the job run should occur in the same session that the procedure was invoked from. The job always runs as the job owner, in the job owner's schema, unless it has credential specified, then the job runs using the user named in the credential.</p> <p>When <code>use_current_session</code> is set to <code>TRUE</code>:</p> <ul style="list-style-type: none"> You can test a job and see any possible errors on the command line. <code>state</code>, <code>run_count</code>, <code>last_start_date</code>, <code>last_run_duration</code>, and <code>failure_count</code> of <code>*_scheduler_jobs</code> are not updated. <code>RUN_JOB</code> can be run in parallel with a regularly scheduled job run. <p>When <code>use_current_session</code> is set to <code>FALSE</code>:</p> <ul style="list-style-type: none"> You need to check the job log to find error information. All relevant fields in <code>*_scheduler_jobs</code> are updated. <code>RUN_JOB</code> fails if a regularly scheduled job is running. <p>For jobs that have a specified destination or destination group, or point to chains or programs with the detached attribute set to <code>TRUE</code>, <code>use_current_session</code> must be <code>FALSE</code>.</p>

Usage Notes

Jobs do not have to be enabled. If a job is disabled, the following validity checks are performed before running it:

- The job points to a valid job class.
- The job owner has `EXECUTE` privileges on the job class.
- If a program or chain is referenced, the program/chain exists.
- If a program or chain is referenced, the job owner has privileges to execute the program/chain.
- All argument values have been set (or have defaults).
- The job owner has the `CREATE EXTERNAL JOB` privilege if this is an external job.

A **TRUE** value for `use_current_session` is not permitted for the following types of jobs:

- Jobs that specify a destination or destination group in the `destination_name` attribute
- Jobs that point to chains (chain jobs)
- Jobs that make use of detached programs (detached jobs).

above bug fix 1261887 6.12.11

When `use_current_session` is **TRUE**, the call to `RUN_JOB` blocks until the job completes. Any errors that occur during the execution of the job are returned as errors to the `RUN_JOB` procedure.

Using `RUN_JOB` with `use_current_session=TRUE` does not update the job state and the job will not appear in `*_SCHEDULER_RUNNING_JOBS` views.

above bug fix 19185117 9.15.14

When `use_current_session` is **FALSE**, `RUN_JOB` returns immediately, and the job is picked up by the job coordinator process and passed on to a job secondary process for execution. The Scheduler views and logs must be queried for the outcome of the job.

Multiple user sessions can use `RUN_JOB` in their sessions simultaneously when `use_current_session` is set to **TRUE**.

`RUN_JOB` requires that you own the job or have **ALTER** privileges on that job. You can also run a job if you have the **CREATE ANY JOB** privilege.

Example

The following is an example of using `RUN_JOB`.

```
BEGIN
  DBMS_SCHEDULER.RUN_JOB(
    JOB_NAME      => 'EODJOB, DSS.ETLJOB',
    USE_CURRENT_SESSION => FALSE);
END;
```

SET_AGENT_REGISTRATION_PASS Procedure

This procedure sets the agent registration password for a database.

A Scheduler agent must register with the database before the database can submit jobs to the agent. The agent must provide this password when registering.

Syntax

```
DBMS_SCHEDULER.SET_AGENT_REGISTRATION_PASS (
  registration_password  IN VARCHAR2,
  expiration_date        IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
  max_uses               IN NUMBER DEFAULT NULL);
```

Parameters

Table 174-81 SET_AGENT_REGISTRATION_PASS Procedure Parameters

Parameter	Description
registration_password	This is the password that remote agents must specify in order to successfully register with the database. If this is NULL, then no agents will be able to register with the database.
expiration_date	If this is set to a non-NULL value, then the registration_password is not valid after this date. After this date, no agents can register with the database. This cannot be set to a date in the past.
max_uses	This is the maximum number of successful registrations that can be performed with this password. After the number of successful registrations has been performed with this password, then no agents can register with the database. This cannot be set to 0 or a negative value. If this is set to NULL, then there will be no limit on the number of successful registrations.

Usage Notes

To prevent abuse, this password can be set to expire after a given date or a maximum number of successful registrations. This procedure will overwrite any password already set. This requires the `MANAGE SCHEDULER` system privilege.

By default, `max_uses` is set to NULL, which means that there is no limit to the number of successful registrations.

Oracle recommends that an agent registration password be reset after every agent registration or every known set of agent registrations. Furthermore, Oracle recommends that this password be set to NULL if no new agents are being registered.

SET_ATTRIBUTE Procedure

This procedure modifies an attribute of a Scheduler object. It is overloaded to accept values of various types.

To set an attribute to NULL, use the `SET_ATTRIBUTE_NULL` procedure. The attributes that can be set depend on the object being altered. All object attributes can be changed, except the object name.

Syntax

```
DBMS_SCHEDULER.SET_ATTRIBUTE (
    name          IN VARCHAR2,
    attribute      IN VARCHAR2,
    value         IN {BOOLEAN|DATE|TIMESTAMP|
                     TIMESTAMP WITH TIME ZONE|TIMESTAMP WITH LOCAL TIME ZONE|
                     INTERVAL DAY TO SECOND});
```

```
DBMS_SCHEDULER.SET_ATTRIBUTE (
    name          IN VARCHAR2,
    attribute      IN VARCHAR2,
    value         IN VARCHAR2,
    value2        IN VARCHAR2 DEFAULT NULL);
```


Parameters

Table 174-82 SET_ATTRIBUTE Procedure Parameters

Parameter	Description
name	The name of the object.
attribute	See Table 174-84 through Table 174-94 .
value	The new value being set for the attribute. This cannot be NULL. To set an attribute value to NULL, use the SET_ATTRIBUTE_NULL procedure.
value2	The value2 argument is for an optional second value. Most attributes have only one value associated with them, but some can have two.

[Table 174-83](#) is a directory of Scheduler object types and tables of attributes for the object types.

These object types can be viewed with Scheduler Data Dictionary Views, listed in *Oracle Database Administrator's Guide*.

Table 174-83 Attribute Tables for Scheduler Object Types

Scheduler Object Type	Table of Attributes
Job	Table 174-84
Program	Table 174-86
Schedule	Table 174-87
File Watcher	Table 174-88
Job Class	Table 174-89
Window	Table 174-90
Chain	Table 174-91
Database Destination	Table 174-92
External Destination	Table 174-93
Group	Table 174-94
Credential	Table 174-95
Resource	Table 174-96

Usage Notes

If an object is altered and it was in the enabled state, the Scheduler first disables it, then makes the change and reenables it. If any errors are encountered during the enable process, the object is not reenabled and an error is generated.

If an object is altered and it was in the disabled state, it remains disabled after it is altered.

To run SET_ATTRIBUTE for a window, a group of type WINDOW, or job class, you must have the MANAGE_SCHEDULER privilege. Otherwise, you must be the owner of the object being altered or have ALTER privileges on that object or have the CREATE ANY JOB privilege.

Job

If there is a running instance of the job when the `SET_ATTRIBUTE` call is made, it is not affected by the call. The change is only affects future runs of the job.

If any of the schedule attributes of a job are altered while the job is running, the time of the next job run is scheduled using the new schedule attributes. Schedule attributes of a job include `schedule_name`, `start_date`, `end_date`, and `repeat_interval`.

If any of the program attributes of a job are altered while the job is running, the new program attributes take effect the next time the job runs. Program attributes of a job include `program_name`, `job_action`, `job_type`, and `number_of_arguments`.

If any job argument values are altered while the job is running, the new values take effect the next time the job runs.

Granting the `ALTER` privilege on a job lets a user alter all attributes of that job except its program attributes (`program_name`, `job_type`, `job_action`, `program_action`, and `number_of_arguments`) and does not allow a user to use a PL/SQL expression to specify the schedule for a job.

Oracle recommends that you not alter a job that was automatically created for you by the database. Jobs that were created by the database have the column `SYSTEM` set to `TRUE` in job views.

Program

If any currently running jobs use the program that was altered, they continue to run with the program definition prior to the alter. The job runs with the new program definition the next time the job executes.

Schedule

If a schedule is altered, the change does not affect running jobs and open windows that use this schedule. The change only goes into effect the next time the jobs runs or the window opens.

File Watcher

If a file watcher is altered, any currently running event-based jobs started by the file arrival event are not affected. On the local system, the new file watcher attributes take effect the next time that the file watcher checks for the arrival of the file (every ten minutes by default). On remote systems, there may be an additional delay before the new file watcher attributes take effect.

Job Class

With the exception of the default job class, all job classes can be altered. To alter a job class, you must have the `MANAGE_SCHEDULER` privilege.

When a job class is altered, running jobs that belong to the class are not affected. The change only takes effect for jobs that have not started running yet. Job Class names must be preceded by `SYS`.

Window

When a window is altered, it does not affect an active window. The changes only take effect the next time the window opens.

If there is no current resource plan, when a window with a designated resource plan opens, the Resource Manager activates with that plan. Window names must be preceded by `SYS`.

Job Attribute Values

Table 174-84 lists attribute values for jobs.



Note:

See the `CREATE_JOB` procedure and the `CREATE_JOBS` procedure for more complete descriptions of the attributes in this table.

Table 174-84 Job Attribute Values

Name	Description
<code>allow_runs_in_restricted_mode</code>	If TRUE, the job is permitted to run when the database is in restricted mode, provided that the job owner is permitted to log in during this mode. FALSE by default.
<code>auto_drop</code>	<p>This attribute, if TRUE, causes a job to be automatically dropped after it completes or is automatically disabled. A job is considered completed if:</p> <ul style="list-style-type: none"> • Its end date (or the end date of the schedule) has passed. • It has run <code>max_runs</code> number of times. <code>max_runs</code> must be set with <code>SET_ATTRIBUTE</code>. • It is not a repeating job and has run once. <p>A job is automatically disabled when it has failed <code>max_failures</code> times. <code>max_failures</code> is also set with <code>SET_ATTRIBUTE</code>.</p> <p>If this attribute is set to FALSE, the jobs are not dropped and their metadata is kept until the job is explicitly dropped with the <code>DROP_JOB</code> procedure.</p> <p>By default, jobs are created with <code>auto_drop</code> set to TRUE.</p>
<code>comments</code>	An optional comment.
<code>connect_credential_name</code>	<p>This attribute may be set to point to a database credential. For a SQL*Plus or backup script job, the credential connects to the database before running the script. For other job types, it is ignored. The job owner must have execute privileges on the credential, otherwise the job fails.</p> <p>Using a <code>connect_credential_name</code> is recommended since it allows the password to be stored securely in a credential in the database rather than in plain view in the job, program action, or script.</p>
<code>credential_name</code>	<p>This attribute specifies the name of the credential object (credential) to use for a remote database job, a remote external job, a local external job, or an event-based job that processes a file arrival event. For local external jobs only, if this attribute is NULL (the default), then a preferred (default) credential is selected. See <i>Oracle Database Administrator's Guide</i> for information about preferred credentials for local external jobs.</p>
<code>database_role</code>	<p>This attribute applies when the database participates in an Oracle Data Guard environment. If this attribute is set to 'PRIMARY', the job runs only when the database is in the role of the primary database. If set to 'LOGICAL STANDBY', the job runs only when the database is in the role of a logical standby. The default is 'PRIMARY' when the database is the primary database, and 'LOGICAL STANDBY' when the database is a logical standby.</p> <p>Note: If you want a job to run for all database roles on a particular host, you must create two copies of the job on that host: one with a <code>database_role</code> of 'PRIMARY', and the other with a <code>database_role</code> of 'LOGICAL STANDBY'.</p>

Table 174-84 (Cont.) Job Attribute Values

Name	Description
destination	<p>*** Deprecated in Oracle Database 11g Release 2. Use destination_name instead.</p> <p>This attribute specifies a host on which to run a remote external job. It must be set to the host name or IP address of the destination host. It can optionally be followed by a port number, in the following format:</p> <p><i>hostname:port</i></p> <p>This attribute is set to NULL by default.</p>
destination_name	<p>The database destination or external destination for the job. Use for remote database jobs and remote external jobs only. For jobs running on the local database or for local external jobs (executables), must be NULL.</p> <p>See Table 174-28 for details about this attribute.</p>
end_date	<p>Specifies the date and time after which the job expires and is no longer run. After the end_date, if is TRUE, the job is dropped. If auto_drop is FALSE, the job is disabled and the STATE of the job is set to COMPLETED.</p> <p>If no value for end_date is specified, the job repeats forever unless max_runs or max_failures is set, in which case the job stops when either value is reached.</p> <p>The value for end_date must be after the value for start_date. If end_date is less than start_date, then an error will be generated. If end_date is the same as start_date, then the job will not execute and no error will be generated.</p>
event_spec	<p>This attribute takes two values: the value argument specifies the event condition and the value2 argument specifies the queue specification. For more details, see the descriptions for the event_condition and queue_spec arguments in the "CREATE_JOB Procedure".</p>
follow_default_timezone	<p>If TRUE and if the job start_date is null, then when the default_timezone scheduler attribute is changed, the Scheduler recomputes the next run date and time for this job so that it is in accordance with the new time zone.</p> <p>For example, if the job was set to run at 02:00 in the previous time zone, it will run at 02:00 in the new time zone.</p> <p>If the job start_date is not null, then the time zone for the run date and time for the job is always specified by the time zone of the start_date.</p> <p>If FALSE, the next start date and time for the job is not recomputed when the default_timezone scheduler attribute is changed. In this case, if the old time zone is three hours earlier than the new time zone, then a job scheduled to run at 02:00 in the old time zone runs at 05:00 in the new time zone.</p> <p>Summer and winter transitions do not change the default time zone name.</p>
instance_id	<p>Valid only in an Oracle Real Application Clusters environment. Indicates the instance on which the job is to be run.</p>

Table 174-84 (Cont.) Job Attribute Values

Name	Description
instance_stickiness	<p>This attribute should only be used for a database running in an Oracle Real Application Clusters (Oracle RAC) environment. By default, it is set to <code>TRUE</code>. If you set <code>instance_stickiness</code> to <code>TRUE</code>, jobs start running on the instance with the lightest load and the Scheduler thereafter attempts to run on the instance that it last ran on. If that instance is either down or so overloaded that it does not start new jobs for a significant period of time, another instance runs the job. If the interval between runs is large, <code>instance_stickiness</code> is ignored and the job is handled as if it were a non-sticky job.</p> <p>If <code>instance_stickiness</code> is set to <code>FALSE</code>, each instance of the job runs on the first instance available.</p> <p>For environments other than Oracle RAC, this attribute is not useful because there is only one instance.</p>
job_action	<p>The action that the job performs, depending on the <code>job_type</code> attribute. For example, if <code>job_type</code> is <code>'STORED_PROCEDURE'</code>, <code>job_action</code> contains the name of the stored procedure.</p>
job_class	<p>The class this job is associated with.</p>
job_priority	<p>This attribute specifies the priority of this job relative to other jobs in the same class as this job. If multiple jobs within a class are scheduled to be executed at the same time, the job priority determines the order in which jobs from that class are picked up for execution by the job coordinator. It can be a value from 1 through 5, with 1 being the first to be picked up for job execution.</p> <p>If no job priority is specified when creating a job, the default priority of 3 is assigned to it.</p>
job_type	<p>The type of this job. Valid values are: <code>'PLSQL_BLOCK'</code>, <code>'STORED_PROCEDURE'</code>, <code>'EXECUTABLE'</code>, <code>CHAIN</code>, <code>'EXTERNAL_SCRIPT'</code>, <code>'SQL_SCRIPT'</code>, and <code>'BACKUP_SCRIPT'</code>.</p> <p>If this is set, <code>program_name</code> must be <code>NULL</code>.</p>
job_weight	<p>*** Deprecated in Oracle Database 11gR2. Do not change the value of this attribute from the default, which is 1.</p> <p>Weight of the job for parallel execution.</p>
logging_level	<p>This attribute specifies how much information is logged. The possible options are:</p> <p><code>DBMS_SCHEDULER.LOGGING_OFF</code></p> <p>(The default) No logging is performed for this job. However, the logging level of the job class takes precedence and job logging may occur.</p> <p><code>DBMS_SCHEDULER.LOGGING_FAILED_RUNS</code></p> <p>The Scheduler logs only jobs that failed, with the reason for failure. If the job class has a higher logging level, then the higher logging level takes precedence.</p> <p><code>DBMS_SCHEDULER.LOGGING_RUNS</code></p> <p>The Scheduler writes detailed information to the job log for all runs of each job in this class. If the job class has a higher logging level, then the higher logging level takes precedence.</p> <p><code>DBMS_SCHEDULER.LOGGING_FULL</code></p> <p>In addition to recording every run of a job, the Scheduler records all operations performed on the job, including create, enable, disable, alter (with <code>SET_ATTRIBUTE</code>), stop, and so on.</p>

Table 174-84 (Cont.) Job Attribute Values

Name	Description
<code>max_failures</code>	<p>This attribute specifies the number of times a job can fail on consecutive scheduled runs before it is automatically disabled. Once a job is disabled, it is no longer executed and its <code>STATE</code> is set to <code>BROKEN</code> in the <code>*_SCHEDULER_JOB</code> views.</p> <p><code>max_failures</code> can be an integer between 1 to 1,000,000. By default, it is set to <code>NULL</code>, which indicates that new instances of the job are started regardless of how many previous instances have failed.</p>
<code>max_run_duration</code>	<p>This attribute specifies the maximum amount of time that the job should be allowed to run. Its datatype is <code>INTERVAL DAY TO SECOND</code>. If this attribute is set to a non-zero and non-<code>NULL</code> value, and job duration exceeds this value, the Scheduler raises an event of type <code>JOB_OVER_MAX_DUR</code>. It is then up to your event handler to decide whether or not to allow the job to continue.</p>
<code>max_runs</code>	<p>This attribute specifies the maximum number of consecutive scheduled runs of the job. Once <code>max_runs</code> is reached, the job is disabled and its state is changed to <code>COMPLETED</code>.</p> <p><code>max_runs</code> can be an integer between 1 and 1,000,000. By default, it is set to <code>NULL</code>, which means that it repeats forever or until <code>end_date</code> or <code>max_failures</code> is reached.</p>
<code>number_of_arguments</code>	<p>The number of arguments if the program is inlined. If this is set, <code>program_name</code> should be <code>NULL</code>.</p>
<code>parallel_instances</code>	<p>This is a boolean attribute that can be set only for event-based jobs.</p> <p>If <code>FALSE</code> (the default), then if an event is raised and the event-based job that processes that event is already running, the new event is ignored.</p> <p>If <code>TRUE</code>, then an instance of the job is started for every instance of the event, and each job instance is a lightweight job so multiple instances of the same event-based job can run in parallel. Each lightweight job takes its attributes (such as action, maximum run duration, and so on) from the definition of the event-based job (its <i>parent job</i>). After the lightweight job completes, it is dropped. There is no explicit limit to the number of lightweight jobs that can run simultaneously to process multiple instances of the event. However, limitations may be imposed by available system resources.</p> <p>The lightweight jobs are not visible in any of the <code>*_SCHEDULER_JOBS</code> views. However, they are visible in the <code>*_SCHEDULER_RUNNING_JOBS</code> views. The name of each lightweight job is the same as that of the parent job, and a subname is automatically generated to distinguish each lightweight job from its parent and from its siblings.</p>
<code>program_name</code>	<p>The name of a program object to use with this job. If this is set, <code>job_action</code>, <code>job_type</code> and <code>number_of_arguments</code> should be <code>NULL</code>.</p>

Table 174-84 (Cont.) Job Attribute Values

Name	Description
raise_events	<p>This attribute tells the Scheduler at what stages of the job execution to raise events. It is a bit vector in which zero or more of the following bits can be set. Each bit has a package constant corresponding to it.</p> <ul style="list-style-type: none"> • job_started CONSTANT PLS_INTEGER := 1 • job_succeeded CONSTANT PLS_INTEGER := 2 • job_failed CONSTANT PLS_INTEGER :=4 • job_broken CONSTANT PLS_INTEGER :=8 • job_completed CONSTANT PLS_INTEGER :=16 • job_stopped CONSTANT PLS_INTEGER :=32 • job_sch_lim_reached CONSTANT PLS_INTEGER :=64 • job_disabled CONSTANT PLS_INTEGER :=128 • job_chain_stalled CONSTANT PLS_INTEGER :=256 • job_all_events CONSTANT PLS_INTEGER := 511 • job_run_completed CONSTANT PLS_INTEGER := job_succeeded + job_failed + job_stopped <p>Table 174-85 describes these event types in detail.</p>
repeat_interval	<p>Either a PL/SQL function returning the next date and time on which to run, or calendaring syntax expression. If this is set, schedule_name should be NULL. See "Calendaring Syntax" for more information.</p>
restartable	<p>This attribute specifies whether or not a job can be restarted in case of failure. By default, jobs are not restartable and this attribute is set to FALSE. Setting this to TRUE means that if a job fails while running, it is restarted from the beginning point of the job.</p> <p>In the case of a chain job, if this attribute is TRUE, the chain is restarted from the beginning after an application failure. If this attribute is FALSE, or if there has been a database failure, the chain is restarted at the last running step. The restart_on_recovery attribute of that step then determines if the step is restarted or marked as stopped. (If marked as stopped, the chain evaluates rules and continues.)</p> <p>Note that setting this attribute to TRUE might lead to data inconsistencies in some situations, for example, if data is committed within a job.</p> <p>Retries on errors are not counted as regular runs. The run count or failure count is not incremented until the job succeeds or has failed all its six retries.</p> <p>The restartable attribute is used by the Scheduler to determine whether to retry the job not only on regular application errors, but after a database malfunction as well. The Scheduler retries the job a maximum of six times. The first time, it waits for one second and multiplies this wait time with a factor of 10 each time thereafter.</p> <p>Both the run count and failure count are incremented by 1 if the job has failed all its six retries. If the job immediately succeeds, or it succeeds on one of its retries, run count is incremented by 1.</p> <p>The Scheduler stops retrying a job when:</p> <ul style="list-style-type: none"> • One of the retries succeeds. • All of its six retries have failed. • The next retry would occur after the next regularly scheduled run of the job. <p>The Scheduler no longer retries the job if the next scheduled retry is past the next regularly scheduled run for repeating jobs.</p>

Table 174-84 (Cont.) Job Attribute Values

Name	Description
<code>schedule_limit</code>	<p>In heavily loaded systems, jobs are not always started at their scheduled time. This attribute enables you to have the Scheduler not start a job at all if the delay in starting the job is larger than the interval specified. It can be a value of 1 minute to 99 days. For example, if a job was supposed to start at noon and the schedule limit is set to 60 minutes, the job will not be run if it has not started to run by 1:00 p.m.</p> <p>If <code>schedule_limit</code> is not specified, the job is executed at some later date as soon as there are resources available to run it. By default, this attribute is set to null, which indicates that the job can be run at any time after its scheduled time. A scheduled job run that is skipped because of this attribute does not count against the number of runs and failures of the job. An entry in the job log reflects the skipped run.</p>
<code>schedule_name</code>	The name of a schedule, window, or group of type <code>WINDOW</code> to use as the schedule for this job. If this is set, <code>end_date</code> , <code>start_date</code> and <code>repeat_interval</code> should all be <code>NULL</code> .
<code>start_date</code>	The original date and time on which this job started or is scheduled to start. If this is set, <code>schedule_name</code> should be <code>NULL</code> .
<code>stop_on_window_close</code>	<p>This attribute only applies if the schedule of a job is a window or a window group. Setting this attribute to <code>TRUE</code> implies that the job should stop once the associated window is closed. The job is stopped using the <code>stop_job</code> procedure with force set to <code>FALSE</code>.</p> <p>By default, <code>stop_on_window_close</code> is set to <code>FALSE</code>. Therefore, if you do not set this attribute, the job continues after the window closes.</p> <p>Note that, although the job is allowed to continue, its resource allocation will probably change because closing a window generally also implies a change in resource plans.</p>
<code>store_output</code>	This is a boolean attribute. If set to <code>TRUE</code> , then for job runs that are logged, all job output and error messages are stored in the <code>*_JOB_RUN_DETAILS</code> views. If set to <code>FALSE</code> , then the output and messages are not stored. For new jobs, this is set, by default, to <code>TRUE</code> .

The following event types are valid values for the `raise_events` attribute in [Table 174-84](#).

Table 174-85 Event Types Raised by the Scheduler

Event Type	Description
<code>job_all_events</code>	Not an event, but a constant that provides an easy way for you to enable all events
<code>job_broken</code>	The job has been disabled and has changed to the <code>BROKEN</code> state because it exceeded the number of failures defined by the <code>max_failures</code> job attribute
<code>job_chain_stalled</code>	A job running a chain is in the <code>CHAIN_STALLED</code> state. A running chain becomes stalled if there are no steps running or scheduled to run and the chain <code>evaluation_interval</code> is set to <code>NULL</code> . No progress is made in the chain unless there is manual intervention.
<code>job_completed</code>	The job completed because it reached its <code>max_runs</code> or <code>end_date</code>
<code>job_disabled</code>	The job was disabled by the Scheduler or by a call to <code>SET_ATTRIBUTE</code>
<code>job_failed</code>	The job failed, either due to an error or an unusual termination.

Table 174-85 (Cont.) Event Types Raised by the Scheduler

Event Type	Description
job_over_max_dur	The job exceeded the maximum run duration specified by its <code>max_run_duration</code> attribute. (Note: you do not need to enable this event with the <code>raise_events</code> job attribute; it is always enabled.)
job_run_completed	A job run either failed, succeeded, or was stopped
job_sch_lim_reached	The schedule limit of the job was reached. The job was not started because the delay in starting the job exceeded the value of the <code>schedule_limit</code> job attribute.
job_started	The job started
job_stopped	The job was stopped by a call to <code>STOP_JOB</code>
job_succeeded	The job completed successfully

Program Attribute Values

Table 174-86 lists program attribute values.

**Note:**

See the "[CREATE_PROGRAM Procedure](#)" for more complete descriptions of the attributes in this table.

Table 174-86 Program Attribute Values

Name	Description
comments	An optional comment. This can describe what the program does or give usage details.
detached	If <code>TRUE</code> , the program is a detached program. See <i>Oracle Database Administrator's Guide</i> for information about detached jobs and detached programs.
number_of_arguments	The number of arguments required by the stored procedure or other executable that the program invokes
program_action	The action that the program performs, indicated by the <code>program_type</code> attribute. For example, if <code>program_type</code> is <code>'STORED_PROCEDURE'</code> , <code>program_action</code> contains the name of the stored procedure.
program_type	The type of program. This must be one of these supported program types: <code>'PLSQL_BLOCK'</code> , <code>'STORED_PROCEDURE'</code> , and <code>'EXECUTABLE'</code> .

Schedule Attribute Values

Table 174-87 lists schedule attribute values.

**Note:**

See "[CREATE_SCHEDULE Procedure](#)" for more complete descriptions of the attributes in this table.

Table 174-87 Schedule Attribute Values

Name	Description
comments	An optional comment.
end_date	The cutoff date and time after which the schedule does not specify any dates.
event_spec	This attribute takes two values: the <code>value</code> argument should contain the event condition and the <code>value2</code> argument should contain the queue specification. For more details, see the descriptions for the <code>event_condition</code> and <code>queue_spec</code> arguments to the " CREATE_JOB Procedure ".
repeat_interval	An attribute specifying how often the schedule should repeat, using the calendaring syntax. See " Calendaring Syntax " for more information.
start_date	The start or reference date and time used by the calendaring syntax.

File Watcher Attribute Values

[Table 174-88](#) lists file watcher attribute values.

Table 174-88 File Watcher Attribute Values

Parameter	Description
destination	Remote host name or IP address where the file is expected to arrive. If <code>NULL</code> , destination is the local host.
directory_path	Directory in which the file is expected to arrive. The single wildcard '?' at the beginning of the path denotes the Oracle home path. For example, '?/rdbms/log' denotes the rdbms/log subdirectory of the Oracle home directory.
file_name	Name of the file being looked for. Two wildcards are permitted anywhere in the file name: '?' denotes any single character, and '*' denotes zero or more characters. This attribute cannot be <code>NULL</code> .
credential_name	Name of a valid credential object. The file watcher uses the credential to authenticate itself with the host operating system to access the watched-for file. The file watcher owner must have the <code>EXECUTE</code> privilege on the credential. Cannot be <code>NULL</code> .
min_file_size	Minimum file size in bytes before the file watcher considers the file found. Default is 0.
steady_state_duration	Minimum time interval that the file must remain unchanged before the file watcher considers the file found. If <code>NULL</code> , an internal value is used. The lower limit for this attribute is 10 seconds.
comments	Optional comment.

Job Class Attribute Values

[Table 174-89](#) lists job class attribute values.

**Note:**

See the "[CREATE_JOB_CLASS Procedure](#)" for more complete descriptions of the attributes in this table.

Table 174-89 Job Class Attribute Values

Name	Description
comments	An optional comment about the class.
log_history	This attribute controls the number of days that job log entries for jobs in this class are retained. It helps prevent the job log from growing indiscriminately. The range of valid values is 0 through 1000000. If set to 0, no history is kept. If NULL, retention days are set by the log_history Scheduler attribute (set with SET_SCHEDULER_ATTRIBUTE).
logging_level	This attribute specifies how much information is logged. The valid values are: <ul style="list-style-type: none"> DBMS_SCHEDULER.LOGGING_OFF No logging is performed for any jobs in this class. DBMS_SCHEDULER.LOGGING_FAILED_RUNS The Scheduler logs only jobs in the class that failed, with the reason for failure. DBMS_SCHEDULER.LOGGING_RUNS The Scheduler writes detailed information to the job log for all runs of each job in this class. This is the default. DBMS_SCHEDULER.LOGGING_FULL The Scheduler records all operations performed on all jobs in this class, in addition to recording every run of a job. Every time a job is created, enabled, disabled, altered (with SET_ATTRIBUTE), stopped, and so on, an entry is recorded in the log.
resource_consumer_group	The resource consumer group that a class is associated with. All jobs in the class run under this resource consumer group. See <i>Oracle Database Administrator's Guide</i> for a description of resource consumer groups and the Database Resource Manager.
service	The database service that the jobs in the job class have affinity to. If both the resource_consumer_group and service attributes are set for a job class, and if the service is mapped to a resource consumer group, the resource_consumer_group attribute takes precedence.

Window Attribute Values

[Table 174-90](#) lists window attribute values.

**Note:**

See the "[CREATE_WINDOW Procedure](#)" for more complete descriptions of the attributes in this table.

Table 174-90 Window Attribute Values

Name	Description
comments	An optional comment about the window.
duration	The duration of the window.
end_date	The date after which the window no longer opens. If this is set, <code>schedule_name</code> must be NULL.
repeat_interval	An attribute specifying how often the schedule should repeat, using the calendaring syntax. PL/SQL date functions are not allowed. If this is set, <code>schedule_name</code> must be NULL. See "Calendaring Syntax" for more information.
resource_plan	<p>The resource plan to be associated with a window. When the window opens, the system switches to this resource plan. When the window closes, the original resource plan is restored. If a resource plan has been made active with the <code>force</code> option, no resource plan switch occurs.</p> <p>Only one resource plan can be associated with a window. It may be NULL or the empty string (""). When it is NULL, the resource plan that is in effect when the window opens stays in effect for the duration of the window. When it is the empty string, the resource manager is disabled for the duration of the window.</p>
schedule_name	The name of a schedule to use with this window. If this is set, <code>start_date</code> , <code>end_date</code> , and <code>repeat_interval</code> must all be NULL.
start_date	The next date and time on which this window is scheduled to open. If this is set, <code>schedule_name</code> must be NULL.
window_priority	The priority of the window. Must be either 'LOW' (default) or 'HIGH'.

Chain Attribute Values

[Table 174-91](#) lists chain attribute values.



Note:

See the ["CREATE_CHAIN Procedure"](#) for more complete descriptions of the attributes in this table.

Table 174-91 Chain Attribute Values

Name	Description
comments	An optional comment describing the purpose of the chain.
evaluation_interval	<p>If not NULL, provides an additional evaluation of the chain at this interval, as well as at normal evaluation times (when the job starts, when a step completes, or when an event that is associated with an event step arrives)</p> <p>This attribute should only be used when chain rules use SQL syntax and the rule conditions contain elements that are not under the control of the Scheduler, because the extra interval is CPU intensive. For most chains, the normal evaluation times are sufficient.</p>

Table 174-91 (Cont.) Chain Attribute Values

Name	Description
rule_set_name	<p>In the normal case, no rule set should be passed in. The Scheduler automatically creates a rule set and associated empty evaluation context. You then use <code>DEFINE_CHAIN_RULE</code> to add rules and <code>DROP_CHAIN_RULE</code> to remove them.</p> <p>Advanced users can create a rule set that describes their chain dependencies and pass it in here. This allows greater flexibility in defining rules. For example, conditions can refer to external variables, and tables can be exposed through the evaluation context. If you pass in a rule set, you must ensure that it is in the format of a chain rule set. (For example, all steps must be listed as variables in the evaluation context). If no rule set is passed in, the rule set created is of the form <code>SCHED_RULESET\${N}</code> and the evaluation context created is of the form <code>SCHED_EVCTX\${N}</code></p>

Database Destination Attribute Values

Table 174-92 lists database destination attribute values.

**Note:**

See the "[CREATE_DATABASE_DESTINATION Procedure](#)" for more complete descriptions of the attributes in this table.

Table 174-92 Database Destination Attribute Values

Name	Description
agent	<p>The name of the external destination (also known as agent destination) that is used to connect to the remote database.</p> <p>You can obtain valid external destination names from the view <code>ALL_SCHEDULER_EXTERNAL_DESTS</code>.</p>
connect_info	<p>The TNS connect descriptor that identifies the remote database to connect to, or the net service name (alias) in <code>tnsnames.ora</code> that resolves to the connect descriptor.</p> <p>Note: This corresponds to the <code>tns_name</code> argument of <code>CREATE_DATABASE_DESTINATION</code>.</p>
enabled	If TRUE, the database destination is enabled.
comments	An optional comment about the database destination.

External Destination Attribute Values

Table 174-93 lists external destination attribute values.

**Note:**

External destinations are created only implicitly by registering a remote Scheduler agent with the local database.

Table 174-93 External Destination Attribute Values

Name	Description
hostname	(GET_ATTRIBUTE only) The fully qualified host name (including domain) or IP address of the computer on which the Scheduler agent resides.
port	(GET_ATTRIBUTE only) The TCP port number on which the agent listens.
ip_address	(GET_ATTRIBUTE only) The IP address of the host on which the agent resides.
enabled	If TRUE, the external destination is enabled.
comments	An optional comment about the external destination.

Group Attribute Values

Table 174-94 lists group attribute values.



Note:

See the "[CREATE_GROUP Procedure](#)" for more complete descriptions of the attributes in this table.

Table 174-94 Group Attribute Values

Name	Description
group_type	(GET_ATTRIBUTE only) The group type (either WINDOW, DB_DEST, or EXTERNAL_DEST).
member_name	Comma-separated list of members. Replaces the existing list of members. To add one or more members to the existing list, use ADD_GROUP_MEMBER. Note: this attribute corresponds to the <code>member</code> argument of CREATE_GROUP.
enabled	If TRUE, the group is enabled.
comments	An optional comment about the group.
number_of_members	(GET_ATTRIBUTE only) The number of members in the group.

Credential Attribute Values

Table 174-95 lists credential attribute values.



Note:

Credential attribute values for the SET_ATTRIBUTE and GET_ATTRIBUTE procedures are deprecated with Oracle Database Release 12c Release 1 (12.1). While these attribute values remain available in this package, for reasons of backward compatibility, Oracle recommends using the alternative enhanced functionality provided in the [DBMS_CREDENTIAL](#) package, specifically the attribute parameter in the [UPDATE_CREDENTIAL Procedure](#).

Table 174-95 Credential Attribute Values

Name	Description
username	The user name for logging into to the host operating system or remote Oracle database. Maximum length is 64.
password	The password for the user name. Maximum length is 128.
comments	A description of the credential. Maximum length is 240.
windows_domain	For a Windows remote executable target, this is the domain that the specified user belongs to. Maximum length is 64.
database_role	The value of the <code>database_role</code> attribute is used as the system privilege for logging into a remote database to run a remote database job. Valid values are: <code>SYSDBA</code> and <code>SYSOPER</code> .

Resource Attribute Values

[Table 174-96](#) lists resource attribute values.

Table 174-96 Resource Attribute Values

Name	Description
resource_name	The name of the resource
units	The number of units of this resource that the job or program uses.
status	The status of the resource. <code>ENFORCE_CONSTRAINTS</code> . This is the default value, and when set, will force the scheduler to enforce resource limits. When the maximum number of units of this resource has been reached, no additional jobs using this resource will get started. <code>IGNORE_CONSTRAINTS</code> . When set, the scheduler will ignore any constraints on this resource. <code>BLOCKED_ALL_JOBS</code> . No jobs having a constraint on this resource will be allowed to run. The resource is considered to be permanently blocking until switched to one of the other two states.
constraint_level	Level of the constraint: <code>JOB_LEVEL</code> or <code>PROGRAM_LEVEL</code> For incompatibilities, for <code>JOB_LEVEL</code> , the incompatibility members must be jobs; for <code>PROGRAM_LEVEL</code> the incompatibility members must be programs.
comments	Descriptive comment about the resource.

SET_ATTRIBUTE_NULL Procedure

This procedure sets an attribute of an object to `NULL`.

The attributes that can be set depend on the object being altered. If the object is enabled, it is disabled before being altered and reenabled afterward. If the object cannot be reenabled, an error is generated and the object is left in a disabled state.

Syntax

```
DBMS_SCHEDULER.SET_ATTRIBUTE_NULL (
  name          IN VARCHAR2,
  attribute     IN VARCHAR2);
```

Parameters

Table 174-97 SET_ATTRIBUTE_NULL Procedure Parameters

Parameter	Description
name	The name of the object
attribute	The attribute being changed

Usage Notes

To run `SET_ATTRIBUTE_NULL` for a window, group of type `WINDOW`, or job class, you must have the `MANAGE SCHEDULER` privilege. Otherwise, you must be the owner of the object being altered or have `ALTER` privileges on that object or have the `CREATE ANY JOB` privilege.

SET_JOB_ANYDATA_VALUE Procedure

This procedure sets the value for an argument of the associated program for a job, encapsulated in an `AnyData` object.

It overrides any default value set for the program argument. `NULL` is a valid assignment for a program argument.

The argument can be specified by position or by name. You can specify by name only when:

- The job points to a saved program object
- The argument was assigned a name with the [DEFINE_ANYDATA_ARGUMENT Procedure](#)

Scheduler does no type checking of the argument at any time.

`SET_JOB_ANYDATA_VALUE` is overloaded.

Syntax

Sets a program argument by its position.

```
DBMS_SCHEDULER.SET_JOB_ANYDATA_VALUE (  
    job_name           IN VARCHAR2,  
    argument_position  IN PLS_INTEGER,  
    argument_value     IN SYS.ANYDATA);
```

Sets a program argument by its name.

```
DBMS_SCHEDULER.SET_JOB_ANYDATA_VALUE (  
    job_name           IN VARCHAR2,  
    argument_name      IN VARCHAR2,  
    argument_value     IN SYS.ANYDATA);
```

Parameters

Table 174-98 SET_JOB_ANYDATA_VALUE Procedure Parameters

Parameter	Description
job_name	The name of the job to be altered
argument_name	The name of the program argument being set

Table 174-98 (Cont.) SET_JOB_ANYDATA_VALUE Procedure Parameters

Parameter	Description
argument_position	The position of the program argument being set
argument_value	The new value to be assigned to the program argument, encapsulated in an AnyData object

Usage Notes

SET_JOB_ANYDATA_VALUE requires that you own the job or have ALTER privileges on that job. You can also set a job argument value if you have the CREATE ANY JOB privilege.

SET_JOB_ANYDATA_VALUE does not apply to lightweight jobs because lightweight jobs cannot take AnyData arguments.



See Also:

- ["SET_JOB_ARGUMENT_VALUE Procedure"](#)
- ["DEFINE_ANYDATA_ARGUMENT Procedure"](#)

SET_JOB_ARGUMENT_VALUE Procedure

This procedure sets the value of an argument for a job.

It overrides any default value set for the corresponding program or stored procedure argument. The argument can be specified by position or by name. You can specify by name only when:

- The job points to a saved program object
- The argument was assigned a name with the [DEFINE_PROGRAM_ARGUMENT Procedure](#) or the [DEFINE_METADATA_ARGUMENT Procedure](#)

Scheduler does no type checking of the argument at any time.

SET_JOB_ARGUMENT_VALUE is overloaded.

Syntax

Sets an argument value by position:

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (  
    job_name           IN VARCHAR2,  
    argument_position  IN PLS_INTEGER,  
    argument_value     IN VARCHAR2);
```

Sets an argument value by name:

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (  
    job_name           IN VARCHAR2,  
    argument_name      IN VARCHAR2,  
    argument_value     IN VARCHAR2);
```

Parameters

Table 174-99 SET_JOB_ARGUMENT_VALUE Procedure Parameters

Parameter	Description
job_name	The name of the job to be altered
argument_name	The name of the program argument being set
argument_position	The position of the program argument being set
argument_value	The new value to be set for the program argument. To set a non-VARCHAR value, use the SET_JOB_ANYDATA_VALUE procedure.

Usage Notes

SET_JOB_ARGUMENT_VALUE requires that you be the owner of the job or have ALTER privileges on that job. You can also set a job argument value if you have the CREATE ANY JOB privilege.

SET_JOB_ARGUMENT_VALUE only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.

SET_JOB_ARGUMENT_VALUE can be used to set arguments of lightweight jobs but only if the argument is of type VARCHAR2.



See Also:

- ["SET_JOB_ANYDATA_VALUE Procedure"](#)
- ["DEFINE_PROGRAM_ARGUMENT Procedure"](#)

SET_JOB_ATTRIBUTES Procedure

This procedure changes an attribute of a job.

Syntax

```
DBMS_SCHEDULER.SET_JOB_ATTRIBUTES (
    jobattr_array    IN JOBATTR_ARRAY,
    commit_semantics IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters

Table 174-100 SET_JOB_ATTRIBUTES Procedure Parameters

Parameter	Description
jobattr_array	The array of job attribute changes.

Table 174-100 (Cont.) SET_JOB_ATTRIBUTES Procedure Parameters

Parameter	Description
commit_semantics	The commit semantics. The following types are supported: <ul style="list-style-type: none">• <code>STOP_ON_FIRST_ERROR</code> returns on the first error and commits previous successful attribute changes to disk. This is the default.• <code>TRANSACTIONAL</code> returns on the first error and rolls back everything that happened before that error.• <code>ABSORB_ERRORS</code> tries to absorb any errors and complete the rest of the job attribute changes on the list. It commits all the successful changes. If errors occur, you can query the view <code>SCHEDULER_BATCH_ERRORS</code> for details.

Usage Notes

Calling `SET_ATTRIBUTE` on an enabled job disables the job, changes the attribute value, and reenables the job. `SET_JOB_ATTRIBUTES` changes the attribute values in the context of a single transaction.

SET_RESOURCE_CONSTRAINT Procedure

This procedure allows users to specify the resources used by jobs.

Syntax

```
DBMS_SCHEDULER.SET_RESOURCE_CONSTRAINT (  
    object_name      IN VARCHAR2,  
    resource_name    IN VARCHAR2,  
    units            IN NUMBER DEFAULT 1);
```

Parameters

Table 174-101 SET_RESOURCE_CONSTRAINT Procedure Parameters

Parameter	Description
object_name	The name of a program or a job, or a comma separated list of these objects.
resource_name	The name of the resource.
units	The number of units of this resource that the job or program uses.

Usages Notes

`object_name` can be the name or comma-separated list of names of either programs or jobs. This creates a constraint on the named resource for these programs or jobs.

`units` specifies the number of units of the resource that the program or job can use. If `units` is set to 0, then the program or job does not use this resource anymore, and the resulting constraint is deleted. Setting `units` to 0 on a resource with no previous constraint results in an error.

When multiple constraints are defined on the same resource, the object types must match. When one or more existing constraints for a resource are based on jobs and a new constraint

is added for the same resource that is based on a program (or vice versa) an error will be raised.

SET_SCHEDULER_ATTRIBUTE Procedure

This procedure sets the value of a Scheduler attribute. This takes effect immediately but the resulting changes may not be seen immediately, depending on the attribute affected.

[Table 174-102](#) provides short attribute descriptions for the `SET_SCHEDULER_ATTRIBUTE` procedure. For complete descriptions, see section "Setting Scheduler Preferences" in *Oracle Database Administrator's Guide*.

Syntax

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE (
    attribute    IN VARCHAR2,
    value        IN VARCHAR2);
```

Parameters

Table 174-102 SET_SCHEDULER_ATTRIBUTE Procedure Parameters

Parameter	Description
attribute	<p>The name of the Scheduler attribute. Possible values are:</p> <ul style="list-style-type: none"> 'default_timezone': Repeating jobs and windows that use the calendaring syntax retrieve the time zone from this attribute when <code>start_date</code> is not specified. See "Calendaring Syntax" for more information. 'email_server': The SMTP server address that the Scheduler uses to send e-mail notifications for job state events. E-mail notifications cannot be sent if this attribute is <code>NULL</code>. 'email_sender': The default e-mail address of the sender of job state e-mail notifications. 'email_server_credential': The schema and name of an existing credential object that SYS has execute object privileges on. Default is <code>NULL</code>. The username and password stored in this credential are used to authenticate with the e-mail server when sending e-mail notifications. 'email_server_encryption': This attribute indicates whether or not encryption is enabled for this email server connection, and if so, at what point encryption starts, and with which protocol. Values are: <ul style="list-style-type: none"> NONE: the default, indicating no encryption used SSL_TLS: indicating that either SSL or TLS are used, from the beginning of the connection STARTTLS: indicating that the connection starts unencrypted, but the command STARTTLS is sent to the e-mail server and starts encryption 'event_expiry_time': The time, in seconds, before a job state event generated by the Scheduler expires from the Scheduler event queue. If <code>NULL</code>, job state events expire after 24 hours. 'log_history': The number of days that log entries for both the job log and the window log are retained. Default is 30 and the range of valid values is 0 through 1000000. 'max_job_slave_processes': This Scheduler attribute is not used.
value	The new value of the attribute

Usage Notes

To run `SET_SCHEDULER_ATTRIBUTE`, you must have the `MANAGE SCHEDULER` privilege.



See Also:

Oracle Database Administrator's Guide for more detailed descriptions of Scheduler attributes

STOP_JOB Procedure

This procedure stops currently running jobs or all jobs in a job class.

After stopping the job, the state of a one-time job is set to `STOPPED`, whereas the state of a repeating job is set to `SCHEDULED` or `COMPLETED`, depending on whether the next run of the job is scheduled.

If a job pointing to a chain is stopped, all running steps of the running chain are stopped.

If a job has multiple destinations, the database attempts to stop the job at all destinations.

For external jobs, `STOP_JOB` stops only the external process that was directly started by the job action. It does not stop child processes of external jobs.

For in-memory full jobs in an Oracle Real Application Clusters environment, `STOP_JOB` uses the `instance_id` attribute of the job definition to determine in which instance (or all of them if the attribute is left null) to stop the in-memory full job. (In-memory full jobs are kept cached in memory, and as such are limited to the instance currently caching them. Because of this, the same `job_name` can in some conditions be used for different jobs on different instances.)

Syntax

```
DBMS_SCHEDULER.STOP_JOB (  
    job_name          IN VARCHAR2  
    force             IN BOOLEAN DEFAULT FALSE  
    commit_semantics  IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR');
```

Parameters

Table 174-103 STOP_JOB Procedure Parameters

Parameter	Description
job_name	<p>Name of a job to stop. Can be a comma-separate list of jobs, where each entry can be one of the following:</p> <ul style="list-style-type: none"> Job name: the name of an existing job, optionally preceded by a schema name and dot separator. Job destination ID: a number, obtained from the <code>JOB_DEST_ID</code> column of the <code>*_SCHEDULER_JOB_DESTS</code> views, that represents the unique combination of a job, a credential, and a destination. Job class: the name of a job class. Must be preceded by the <code>SYS</code> schema name and a dot separator. <p>If you specify a job class, all jobs that belong to that job class are stopped. If you specify a job that was created with a destination group as its <code>destination_name</code> attribute, all job instances on all destinations are stopped.</p>
force	<p>If <code>force</code> is set to <code>FALSE</code>, the Scheduler tries to gracefully stop the job using an interrupt mechanism. This method gives control back to the secondary process, which can update the status of the job in the job queue to stopped. If this fails, an error is returned.</p> <p>If <code>force</code> is set to <code>TRUE</code>, the Scheduler immediately terminates the secondary process. Oracle recommends that <code>STOP_JOB</code> with <code>force</code> set to <code>TRUE</code> be used only after a <code>STOP_JOB</code> with <code>force</code> set to <code>FALSE</code> has failed.</p> <p>Use of the <code>force</code> option requires the <code>MANAGE SCHEDULER</code> system privilege.</p>
commit_semantics	<p>The commit semantics. The following two types are supported:</p> <ul style="list-style-type: none"> <code>STOP_ON_FIRST_ERROR</code>: The procedure returns on the first error and commits previous successful stop operations to disk. This is the default. <code>ABSORB_ERRORS</code>: The procedure tries to absorb any errors, stops the rest of the jobs, and commits all the successful stop operations. This type is available only if no job classes are specified in the <code>job_name</code> list. If errors occur, you can query the view <code>SCHEDULER_BATCH_ERRORS</code> for details.

Usage Notes

`STOP_JOB` without the `force` option requires that you be the owner of the job or have `ALTER` privileges on that job. You can also stop a job if you have the `CREATE ANY JOB` or `MANAGE SCHEDULER` privilege.

`STOP_JOB` with the `force` option requires that you have the `MANAGE SCHEDULER` privilege.

Example

The following is an example of using `STOP_JOB`.

```
BEGIN
  DBMS_SCHEDULER.STOP_JOB('DSS.ETLJOB, 984, 1223, SYS.ETL_JOBCLASS');
END;
```