29 Administering Oracle Scheduler

You can configure, manage, monitor, and troubleshoot Oracle Scheduler.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

> **Note:**
>
> This chapter describes how to use the `DBMS_SCHEDULER` package to administer Oracle Scheduler. You can accomplish many of the same tasks using Oracle Enterprise Manager Cloud Control.
>
> See *Oracle Database PL/SQL Packages and Types Reference* for `DBMS_SCHEDULER` information and the Cloud Control online help for information on Oracle Scheduler pages.
>
> See *Oracle Multitenant Administrator's Guide* for information on using Oracle Scheduler with CDB.

- Configuring Oracle Scheduler
  Configuring Oracle Scheduler includes tasks such as setting privileges and preferences, and using the Oracle Scheduler agent to run remote jobs.
- Monitoring and Managing the Scheduler
  You can view the currently active window and the resource plan associated with it, view information about currently running jobs, monitor and manage window and job logs, and manage Scheduler security.
- Import/Export and the Scheduler
  You must use the Data Pump utilities (`impdp` and `expdp`) to export Scheduler objects.
- Troubleshooting the Scheduler
  You can troubleshoot problems with Scheduler.
- Examples of Using the Scheduler
  Examples illustrate using Scheduler.
- Scheduler Reference
  There are several privileges and data dictionary views related to Scheduler.

29.1 Configuring Oracle Scheduler

Configuring Oracle Scheduler includes tasks such as setting privileges and preferences, and using the Oracle Scheduler agent to run remote jobs.

- Setting Oracle Scheduler Privileges
  You must have the SCHEDULER_ADMIN role to perform all Oracle Scheduler administration tasks. Typically, database administrators already have this role with the ADMIN option as part of the DBA role.
- Setting Scheduler Preferences
  There are several system-wide Scheduler preferences that you can set. You set these preferences by setting Scheduler attributes with the SET_SCHEDULER_ATTRIBUTE procedure in the DBMS_SCHEDULER package.
- Using the Oracle Scheduler Agent to Run Remote Jobs
  The Oracle Scheduler agent can schedule and run remote jobs.

29.1.1 Setting Oracle Scheduler Privileges

You must have the SCHEDULER_ADMIN role to perform all Oracle Scheduler administration tasks. Typically, database administrators already have this role with the ADMIN option as part of the DBA role.

For example, users SYS and SYSTEM are granted the DBA role. You can grant this role to another administrator by issuing the following statement:

```
GRANT SCHEDULER_ADMIN TO username;
```

Because the SCHEDULER_ADMIN role is a powerful role allowing a grantee to execute code as any user, you should consider granting individual Scheduler system privileges instead. Object and system privileges are granted using regular SQL grant syntax, for example, if the database administrator issues the following statement:

```
GRANT CREATE JOB TO scott;
```

After this statement is executed, scott can create jobs, schedules, programs, and file watchers in their schema. As another example, the database administrator can issue the following statement:

```
GRANT MANAGE SCHEDULER TO adam;
```

After this statement is executed, adam can create, alter, or drop windows, job classes, or window groups. adam will also be able to set and retrieve Scheduler attributes and purge Scheduler logs.

Setting Chain Privileges

Scheduler chains use underlying Oracle Rules Engine objects along with their associated privileges. To create a chain in their own schema, users must have the CREATE JOB privilege in addition to the Rules Engine privileges required to create rules, rule sets, and evaluation contexts in their own schema. These can be granted by issuing the following statement:

```
GRANT CREATE RULE, CREATE RULE SET, CREATE EVALUATION CONTEXT TO user;
```

To create a chain in a different schema, users must have the CREATE ANY JOB privilege in addition to the privileges required to create rules, rule sets, and evaluation contexts in schemas other than their own. These can be granted by issuing the following statement:

```
GRANT CREATE ANY RULE, CREATE ANY RULE SET,
   CREATE ANY EVALUATION CONTEXT TO user;
```

Altering or dropping chains in schemas other than the users's schema require corresponding system Rules Engine privileges for rules, rule sets, and evaluation contexts.

See Also:

"Chain Tasks and Their Procedures" for more information regarding chain privileges.

29.1.2 Setting Scheduler Preferences

There are several system-wide Scheduler preferences that you can set. You set these preferences by setting Scheduler attributes with the SET_SCHEDULER_ATTRIBUTE procedure in the DBMS_SCHEDULER package.

Setting these attributes requires the MANAGE SCHEDULER privilege. The attributes are:

- default_timezone

  It is very important that you set this attribute. Repeating jobs and windows that use the calendaring syntax need to know which time zone to use for their repeat intervals. See "Using the Scheduler Calendaring Syntax". They normally retrieve the time zone from start_date, but if no start_date is provided (which is not uncommon), they retrieve the time zone from the default_timezone Scheduler attribute.

  The Scheduler derives the value of default_timezone from the operating system environment. If the Scheduler can find no compatible value from the operating system, it sets default_timezone to NULL.

  It is crucial that you verify that default_timezone is set properly, and if not, that you set it. To verify it, run this query:

  ```
  SELECT DBMS_SCHEDULER.STIME FROM DUAL;

  STIME
  ---------------------------------------------------------------------------
  28-FEB-12 09.04.10.308959000 PM UTC
  ```

  To ensure that daylight savings adjustments are followed, it is recommended that you set default_timezone to a region name instead of an absolute time zone offset like '-8:00'. For example, if your database resides in Miami, Florida, USA, issue the following statement:

  ```
  DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('default_timezone','US/Eastern');
  ```

  Similarly, if your database resides in Paris, you would set this attribute to 'Europe/Warsaw'. To see a list of valid region names, run this query:

  ```
  SELECT DISTINCT TZNAME FROM V$TIMEZONE_NAMES;
  ```

  If you do not properly set default_timezone, the default time zone for repeating jobs and windows will be the absolute offset retrieved from SYSTIMESTAMP (the time zone of the operating system environment of the PDB), which means that repeating jobs and windows that do not have their `start_date` set will not follow daylight savings adjustments.

- `email_server`

  This attribute specifies an SMTP server address that the Scheduler uses to send e-mail notifications for job state events. It takes the following format:

  ```
  host[:port]
  ```

  where:

  – *`host`* is the host name or IP address of the SMTP server.

  – *`port`* is the TCP port on which the SMTP server listens. If not specified, the default port of 25 is used.

  If this attribute is not specified, set to `NULL`, or set to an invalid SMTP server address, the Scheduler cannot send job state e-mail notifications.

- `email_sender`

  This attribute specifies the default e-mail address of the sender for job state e-mail notifications. It must be a valid e-mail address. If this attribute is not set or set to `NULL`, then job state e-mail notifications that do not specify a sender address do not have a FROM address in the e-mail header.

- `email_server_credential`

  This attribute specifies the schema and name of an existing credential object. The default is `NULL`.

  When an e-mail notification goes out, the Scheduler determines if the `email_server_credential` points to a valid credential object that `SYS` has execute object privileges on. If the SMTP server specified in the `email_server` attribute requires authentication, then the Scheduler uses the user name and password stored in the specified credential object to authenticate with the e-mail server.

  If the `email_server_credential` is specified, then the `email_server` attribute must specify an SMTP server that requires authentication.

  If the `email_server_credential` is not specified, then the Scheduler supports sending notification e-mails through an SMTP server for which authentication is not configured.

- `email_server_encryption`

  This attribute indicates whether encryption is enabled for this SMTP server connection, and if so, at what point encryption starts, and with which protocol.

  Values for `email_server_encryption` are:

  `NONE`: The default, indicates no encryption.

  `SSL_TLS`: Indicates that either `SSL` or `TLS` are used, from the beginning of the connection. The two sides determine which protocol is most secure. This is the most common setting for this parameter.

  `STARTTLS`: Indicates that the connection starts in an unencrypted state, but then the command `STARTTLS` directs the e-mail server to start encryption using `TLS`.

- `event_expiry_time`

  This attribute enables you to set the time in seconds before a job state event generated by the Scheduler expires (is automatically purged from the Scheduler event queue). If `NULL`, job state events expire after 24 hours.

- `log_history`

  This attribute controls the number of days that log entries for both the job log and the window log are retained. It helps prevent logs from growing indiscriminately. The range of valid values is 0 through 1000000. If set to 0, no history is kept. Default value is 30. You can override this value at the job class level by setting a value for the `log_history` attribute of the job class.

See *Oracle Database PL/SQL Packages and Types Reference* for the syntax for the `SET_SCHEDULER_ATTRIBUTE` procedure.

29.1.3 Using the Oracle Scheduler Agent to Run Remote Jobs

The Oracle Scheduler agent can schedule and run remote jobs.

Using the Oracle Scheduler agent, the Scheduler can schedule and run two types of remote jobs:

- Remote database jobs: Remote database jobs must be run through an Oracle Scheduler agent. Oracle recommends that an agent be installed on the same host as the remote database.

  If you intend to run remote database jobs, the Scheduler agent must be Oracle Database 11*g* Release 2 (11.2) or later.
- Remote external jobs: Remote external jobs run on the same host that the Scheduler agent is installed on.

  If you intend to run only remote external jobs, Oracle Database 11*g* Release 1 (11.1) of the Scheduler agent is sufficient.

You must install Scheduler agents on all hosts that remote external jobs will run on. You should install Scheduler agents on all hosts running remote databases that remote database jobs will be run on.

Each database that runs remote jobs requires an initial setup to enable secure communications between databases and remote Scheduler agents, as described in "Setting up Databases for Remote Jobs".

Enabling remote jobs involves the following steps:

1. Enabling and Disabling Databases for Remote Jobs
2. Installing and Configuring the Scheduler Agent on a Remote Host
3. Performing Tasks with the Scheduler Agent

- Enabling and Disabling Databases for Remote Jobs
  You can set up databases for remote jobs and disable databases for remote jobs.
- Installing and Configuring the Scheduler Agent on a Remote Host
  Before you can run remote jobs on a particular host, you must install and configure the Scheduler agent.
- Performing Tasks with the Scheduler Agent
  The Scheduler agent is a standalone program that enables you to schedule and run external and database jobs on remote hosts. You start and stop the Scheduler agent using the `schagent` utility on UNIX and Linux, and the `OracleSchedulerExecutionAgent` service on Windows.

See Also:

- "About Remote External Jobs"
- "Database Jobs" for more information on remote database jobs

29.1.3.1 Enabling and Disabling Databases for Remote Jobs

You can set up databases for remote jobs and disable databases for remote jobs.

- Setting up Databases for Remote Jobs
  Before a database can run jobs using a remote Scheduler agent, the database must be properly configured, and the agent must be registered with the database.
- Disabling Remote Jobs
  You can disable remote jobs on a database by dropping the REMOTE_SCHEDULER_AGENT user.

29.1.3.1.1 Setting up Databases for Remote Jobs

Before a database can run jobs using a remote Scheduler agent, the database must be properly configured, and the agent must be registered with the database.

This section describes the configuration, including the required agent registration password in the database. You will later register the database, as shown in "Registering Scheduler Agents with Databases".

You can limit the number of Scheduler agents that can register, and you can set the password to expire after a specified duration.

Complete the following steps once for each database that creates and runs remote jobs.

To set up a database to create and run remote jobs:

1. Ensure that shared server is enabled.

   See "Enabling Shared Server".

   If several Scheduler agents are being used with the same database, set the value of the SHARED_SERVERS database initialization parameter high enough to avoid errors when all those agents try to work in parallel.

   Note:

   If you are running in multitenant mode, you must unlock the anonymous account in CDB$ROOT.

   Using SQL*Plus, connect to CDB$ROOT as SYS user, and enter the following command:

   ```
   SQL> alter session set container = CDB$ROOT;
   SQL> alter user anonymous account unlock container=current;
   ```

2. Using SQL*Plus, connect to the database (specify pluggable database under multitenant mode) as the SYS user.

3. Enter the following command to verify that the XML DB option is installed:

   ```
   SQL> DESC RESOURCE_VIEW
   ```

   If XML DB is not installed, this command returns an "object does not exist" error.

   > **Note:**
   >
   > If XML DB is not installed, you must install it before continuing.

4. If you are using HTTPS connections, then add a certificate to the database wallet as follows:

   > **Note:**
   >
   > Check that the database is not using the wallet while adding the certificate, or else shutdown the database while adding the certificate and then startup the database.

   a. If you do not have an existing database wallet in the `ORACLE_HOME/admin/$ORACLE_SID/xdb_wallet` directory, then create one using the `orapki` command line utility. For example:

      ```
      orapki wallet create -wallet $ORACLE_HOME/admin/$ORACLE_SID/xdb_wallet -
      pwd wallet_password -auto_login
      ```

   b. Add a certificate to the wallet using the `orapki` command line utility. For example:

      ```
      orapki wallet add -wallet $ORACLE_HOME/admin/$ORACLE_SID/xdb_wallet -dn
      CN=fully_qualified_domain_name -self_signed -pwd wallet_password -
      validity number_of_days -keysize key_size_for_the_certificate(512|1024|
      2048)
      ```

   > **Note:**
   >
   > The use of weaker encryption keys is deprecated in Oracle Database 21c.

   > **See Also:**
   >
   > *Oracle Database Security Guide* for more information about adding certificates to a database wallet using the `orapki` utility

5. Enable HTTP(S) connections to the database as follows:

   a. Determine whether or not the Oracle XML DBM HTTP(S) Server is enabled:

      Run the following command for HTTP connections:

      ```
      SQL> SELECT DBMS_XDB_CONFIG.GETHTTPPORT() FROM DUAL;
      ```

Run the following command for HTTPS connections:

```
SQL> SELECT DBMS_XDB_CONFIG.GETHTTPSPORT() FROM DUAL;
```

If the statement returns 0, then Oracle XML DBM HTTP(S) Server is disabled.

b. Enable Oracle XML DB HTTP(S) Server on a nonzero port by logging in as SYS and run the following commands:

   If you are using HTTP connections:

```
SQL> EXEC DBMS_XDB_CONFIG.SETHTTPPORT (port);
SQL> COMMIT;
```

   If you are using HTTPS connections:

```
SQL> EXEC DBMS_XDB_CONFIG.SETHTTPSPORT (port);
SQL> COMMIT;
```

   where *port* is the TCP port number on which you want the database to listen for HTTP(S) connections.

   port must be an integer between 1 and 65536, and for UNIX and Linux must be greater than 1023. Choose a port number that is not already in use.

   Each pluggable database must use a unique port number so that the scheduler agent can determine the exact pluggable database later during the agent registration procedure.

   Note:

   - This enables HTTP(S) connections on all instances of an Oracle Real Application Clusters database.
   - Oracle Scheduler agent supports HTTPS connections starting with Oracle Database 18c.

6. Run the script prvtrsch.plb with following command:

```
SQL> @?/rdbms/admin/prvtrsch.plb
```

7. Set a registration password for the Scheduler agents using the SET_AGENT_REGISTRATION_PASS procedure.

   The following example sets the agent registration password to mypassword.

```
BEGIN
  DBMS_SCHEDULER.SET_AGENT_REGISTRATION_PASS('mypassword');
END;
/
```

   Note:

   You must have the MANAGE SCHEDULER privilege to set an agent registration password. See *Oracle Database PL/SQL Packages and Types Reference* for more information on the SET_AGENT_REGISTRATION_PASS procedure.

You will do the actual registration further on, in "Registering Scheduler Agents with Databases".

29.1.3.1.2 Disabling Remote Jobs

You can disable remote jobs on a database by dropping the `REMOTE_SCHEDULER_AGENT` user.

To disable remote jobs:

- Submit the following SQL statement:

  ```
  DROP USER REMOTE_SCHEDULER_AGENT CASCADE;
  ```

Registration of new scheduler agents and execution of remote jobs is disabled until you run `prvtrsch.plb` again.

29.1.3.2 Installing and Configuring the Scheduler Agent on a Remote Host

Before you can run remote jobs on a particular host, you must install and configure the Scheduler agent.

After installing and configuring the Scheduler agent, you must register and start the Scheduler agent on the host, described in "Performing Tasks with the Scheduler Agent". The Scheduler agent must also be installed in its own Oracle home.

To install and configure the Scheduler agent on a remote host:

1. Download or retrieve the Scheduler agent software, which is available on the Oracle Database Client media included in the Database Media Pack, and online at:

   `http://www.oracle.com/technology/software/products/database`

2. Ensure that you have first properly set up any database on which you want to register the agent.

   See "Enabling and Disabling Databases for Remote Jobs" for instructions.

3. Log in to the host you want to install the Scheduler agent on. This host runs remote jobs.

   - For Windows, log in as an administrator.
   - For UNIX and Linux, log in as the user that you want the Scheduler agent to run as. This user requires no special privileges.

4. Run the Oracle Universal Installer (OUI) from the installation media for Oracle Database Client.

   - For Windows, run `setup.exe`.
   - For UNIX and Linux, use the following command:

     ```
     /directory_path/runInstaller
     ```

   where *`directory_path`* is the path to the Oracle Database Client installation media.

5. On the Select Installation Type page, select **Custom**, and then click **Next**.
6. On the Select Product Languages page, select the desired languages, and click **Next**.
7. On the Specify Install Location page, enter the path for a new Oracle home for the agent, and then click **Next.**
8. On the Available Product Components page, select **Oracle Scheduler Agent**, and click **Next**.
9. On the Oracle Database Scheduler Agent page:

   a. In the Scheduler Agent Hostname field, enter the host name of the computer that the Scheduler agent is installed on.
   b. In the Scheduler Agent Port Number field, enter the TCP port number that the Scheduler agent is to listen on for connections, or accept the default, and then click **Next**.

      Choose an integer between 1 and 65535. On UNIX and Linux, the number must be greater than 1023. Ensure that the port number is not already in use.

      OUI performs a series of prerequisite checks. If any of the prerequisite checks fail, resolve the problems, and then click **Next**.

10. On the Summary page, click **Finish**.
11. (UNIX and Linux only) When OUI prompts you to run the script `root.sh`, enter the following command as the `root` user:

    ```
    script_path/root.sh
    ```

    The script is located in the directory that you chose for agent installation.

    When the script completes, click **OK** in the Execute Configuration Scripts dialog box.

12. Click **Close** to exit OUI when installation is complete.
13. Use a text editor to review the agent configuration parameter file `schagent.conf`, which is located in the Scheduler agent home directory, and verify the port number in the `PORT=` directive.
14. Ensure that any firewall software on the remote host or any other firewall that protects that host has an exception to accommodate the Scheduler agent.

29.1.3.3 Performing Tasks with the Scheduler Agent

The Scheduler agent is a standalone program that enables you to schedule and run external and database jobs on remote hosts. You start and stop the Scheduler agent using the `schagent` utility on UNIX and Linux, and the `OracleSchedulerExecutionAgent` service on Windows.

- About the schagent Utility
  The executable utility `schagent` performs certain tasks for the agent on Windows, UNIX and Linux.
- Using the Scheduler Agent on Windows
  The Windows Scheduler agent service is automatically created and started during installation. The name of the service ends with `OracleSchedulerExecutionAgent`.
- Starting the Scheduler Agent
  Starting the Scheduler agent enables the host on which it resides to run remote jobs.
- Stopping the Scheduler Agent
  Stopping the Scheduler agent prevents the host on which it resides from running remote jobs.
- Registering Scheduler Agents with Databases
  As soon as you have finished configuring the Scheduler Agent, you can register the Agent on one or more databases that are to run remote jobs.

29.1.3.3.1 About the schagent Utility

The executable utility `schagent` performs certain tasks for the agent on Windows, UNIX and Linux.

The options for `schagent` are indicated in Table 29-1.

Use `schagent` with the appropriate syntax and options as follows:

For example:

UNIX and Linux: *AGENT_HOME*`/bin/schagent -status`

Windows: *AGENT_HOME*`/bin/schagent.exe -status`

Table 29-1 schagent options

Option	Description
`-start`	Starts the Scheduler Agent. *UNIX and Linux only*
`-stop`	Prompts the Scheduler agent to stop all the currently running jobs and then stop execution gracefully. *UNIX and Linux only*
`-abort`	Stops the Scheduler agent forcefully, that is, without stopping jobs first. From Oracle Database 11*g* Release 2 (11.2). *UNIX and Linux only*
`-status`	Returns this information about the Scheduler Agent running locally: version, uptime, total number of jobs run since the agent started, number of jobs currently running, and their descriptions.
`-registerdatabase`	Register the Scheduler agent with the base database or additional databases that are to run remote jobs on the agent's host computer.
`-unregisterdatabase`	Unregister an agent from a database.

29.1.3.3.2 Using the Scheduler Agent on Windows

The Windows Scheduler agent service is automatically created and started during installation. The name of the service ends with `OracleSchedulerExecutionAgent`.

> **Note:**
>
> Do not confuse this service with the `OracleJobScheduler` service, which runs on a Windows computer on which an Oracle database is installed, and manages the running of local external jobs without credentials.

29.1.3.3.3 Starting the Scheduler Agent

Starting the Scheduler agent enables the host on which it resides to run remote jobs.

To start the Scheduler agent:

- Do one of the following:

  - On UNIX and Linux, run the following command:

    ```
    AGENT_HOME/bin/schagent -start
    ```

  - On Windows, start the service whose name ends with `OracleSchedulerExecutionAgent`.

29.1.3.3.4 Stopping the Scheduler Agent

Stopping the Scheduler agent prevents the host on which it resides from running remote jobs.

To stop the Scheduler agent:

- Do one of the following:
  - On UNIX and Linux, run the `schagent` utility with either the `-stop` or `-abort` option as described in Table 29-1:

    ```
    AGENT_HOME/bin/schagent -stop
    ```

  - On Windows, stop the service whose name ends with `OracleSchedulerExecutionAgent`. This is equivalent to the `-abort` option.

29.1.3.3.5 Registering Scheduler Agents with Databases

As soon as you have finished configuring the Scheduler Agent, you can register the Agent on one or more databases that are to run remote jobs.

You can also log in later on and register the agent with additional databases.

1. If you have already logged out, then log in to the host that is running the Scheduler agent, as follows:
   - For Windows, log in as an administrator.
   - For UNIX and Linux, log in as the user with which you installed the Scheduler agent.
2. Use the following command for each database that you want to register the Scheduler agent on:
   - On UNIX and Linux, run this command:

     ```
     AGENT_HOME/bin/schagent -registerdatabase db_host db_http(s)_port
     ```

   - On Windows, run this command:

     ```
     AGENT_HOME/bin/schagent.exe -registerdatabase db_host db_http(s)_port
     ```

   where:
   - *`db_host`* is the host name or IP address of the host on which the database resides. In an Oracle Real Application Clusters environment, you can specify any node.
   - *`db_http(s)_port`* is the port number that the database listens on for HTTP(S) connections. You set this parameter previously in "Enabling and Disabling Databases for Remote Jobs". You can check the port number by submitting the following SQL statement to the database:

     For HTTP connections:

     ```
     SELECT DBMS_XDB_CONFIG.GETHTTPPORT() FROM DUAL;
     ```

     For HTTPS connections:

     ```
     SELECT DBMS_XDB_CONFIG.GETHTTPSPORT() FROM DUAL;
     ```

A port number of 0 means that HTTP(S) connections are disabled.

The agent prompts you to enter the agent registration password that you set in "Enabling and Disabling Databases for Remote Jobs".

Starting with Oracle Database 18c, the agent automatically determines if the port is configured to use HTTP or HTTPS connections. For HTTPS connections, the agent also prompts you to select any untrusted certificate that you want to add as a trusted one.

3. Repeat the previous steps for any additional databases to run remote jobs on the agent's host.

29.2 Monitoring and Managing the Scheduler

You can view the currently active window and the resource plan associated with it, view information about currently running jobs, monitor and manage window and job logs, and manage Scheduler security.

- Viewing the Currently Active Window and Resource Plan
  You can view the currently active window and the plan associated with it by querying the `DBA_SCHEDULER_WINDOWS` view.
- Finding Information About Currently Running Jobs
  You can check the state of a job by querying the `DBA_SCHEDULER_JOBS` view.
- Monitoring and Managing Window and Job Logs
  The Scheduler supports two kinds of logs: the job log and the window log.
- DBMS_SCHEDULER In-Memory Trace
  The DBMS_SCHEDULER In-Memory tracing feature provides a tool for temporarily storing the scheduler trace messages generated during process execution.
- Managing Scheduler Security
  You should grant the appropriate privileges to users based on the Scheduler operations they will perform.

29.2.1 Viewing the Currently Active Window and Resource Plan

You can view the currently active window and the plan associated with it by querying the `DBA_SCHEDULER_WINDOWS` view.

For example, issue the following statement:

```
SELECT WINDOW_NAME, RESOURCE_PLAN FROM DBA_SCHEDULER_WINDOWS
WHERE ACTIVE='TRUE';

WINDOW_NAME                    RESOURCE_PLAN
------------------------------ --------------------------
MY_WINDOW10                    MY_RESOURCEPLAN1
```

If there is no window active, you can view the active resource plan by issuing the following statement:

```
SELECT * FROM V$RSRC_PLAN;
```

29.2.2 Finding Information About Currently Running Jobs

You can check the state of a job by querying the `DBA_SCHEDULER_JOBS` view.

For example, issue the following statement:

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME = 'MY_EMP_JOB1';

JOB_NAME                       STATE
------------------------------ ---------
MY_EMP_JOB1                    DISABLED
```

In this case, you could enable the job using the ENABLE procedure. Table 29-2 shows the valid values for job state.

Table 29-2 Job States

Job State	Description
disabled	The job is disabled.
scheduled	The job is scheduled to be executed.
running	The job is currently running.
completed	The job has completed, and is not scheduled to run again.
stopped	The job was scheduled to run once and was stopped while it was running.
broken	The job is broken.
failed	The job was scheduled to run once and failed.
retry scheduled	The job has failed at least once and a retry has been scheduled to be executed.
succeeded	The job was scheduled to run once and completed successfully.
chain_stalled	The job is of type chain and has no steps running, no steps scheduled to run, and no event steps waiting on an event, and the chain evaluation_interval is set to NULL. No progress will be made in the chain unless there is manual intervention.

You can check the progress of currently running jobs by issuing the following statement:

```
SELECT * FROM ALL_SCHEDULER_RUNNING_JOBS;
```

Note that, for the column CPU_USED to show valid data, the initialization parameter RESOURCE_LIMIT must be set to true.

You can check the status of all jobs at all remote and local destinations by issuing the following statement:

```
SELECT * FROM DBA_SCHEDULER_JOB_DESTS;
```

You can find out information about a job that is part of a running chain by issuing the following statement:

```
SELECT * FROM ALL_SCHEDULER_RUNNING_CHAINS WHERE JOB_NAME='MY_JOB1';
```

You can check whether the job coordinator is running by searching for a process of the form cjqNNN.

> **See Also:**
>
> - "Multiple-Destination Jobs"
> - *Oracle Database Reference* for details regarding the `*_SCHEDULER_RUNNING_JOBS` view
> - *Oracle Database Reference* for details regarding the `*_SCHEDULER_JOBS` view

29.2.3 Monitoring and Managing Window and Job Logs

The Scheduler supports two kinds of logs: the job log and the window log.

- Job Log
  You can view information about job runs, job state changes, and job failures in the job log.
- Window Log
  The window log records operations on windows.
- Purging Logs
  To prevent job and window logs from growing indiscriminately, use the `SET_SCHEDULER_ATTRIBUTE` procedure to specify how much history (in days) to keep.

29.2.3.1 Job Log

You can view information about job runs, job state changes, and job failures in the job log.

The job log is implemented as the following two data dictionary views:

- `*_SCHEDULER_JOB_LOG`
- `*_SCHEDULER_JOB_RUN_DETAILS`

You can control the amount of logging that the Scheduler performs on jobs at both the job class and individual job level. Normally, you control logging at the class level, as this offers you more control over logging for the jobs in the class.

See "Viewing the Job Log" for definitions of the various logging levels and for information about logging level precedence between jobs and their job class. By default, the logging level of job classes is `LOGGING_RUNS`, which causes all job runs to be logged.

You can set the `logging_level` attribute when you create the job class, or you can use the `SET_ATTRIBUTE` procedure to change the logging level at a later time. The following example sets the logging level of jobs in the `myclass1` job class to `LOGGING_FAILED_RUNS`, which means that only failed runs are logged. Note that all job classes are in the `SYS` schema.

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   'sys.myclass1', 'logging_level', DBMS_SCHEDULER.LOGGING_FAILED_RUNS);
END;
/
```

You must be granted the `MANAGE SCHEDULER` privilege to set the logging level of a job class.

See Also:

- "Viewing the Job Log" for more detailed information about the job log and for examples of queries against the job log views
- *Oracle Database Reference* for details on the `*_SCHEDULER_JOB_LOG` view
- *Oracle Database Reference* for details on the `*_SCHEDULER_JOB_RUN_DETAILS` view
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `CREATE_JOB_CLASS` and `SET_ATTRIBUTE` procedures
- "Setting Scheduler Preferences" for information about setting retention for log entries

29.2.3.2 Window Log

The window log records operations on windows.

The Scheduler makes an entry in the window log each time that:

- You create or drop a window
- A window opens
- A window closes
- Windows overlap
- You enable or disable a window

There are no logging levels for window activity logging.

To see the contents of the window log, query the `DBA_SCHEDULER_WINDOW_LOG` view. The following statement shows sample output from this view:

```
SELECT log_id, to_char(log_date, 'DD-MON-YY HH24:MI:SS') timestamp,
   window_name, operation FROM DBA_SCHEDULER_WINDOW_LOG;

    LOG_ID TIMESTAMP            WINDOW_NAME       OPERATION
---------- -------------------- ----------------- --------
         4 10/01/2004 15:29:23  WEEKEND_WINDOW    CREATE
         5 10/01/2004 15:33:01  WEEKEND_WINDOW    UPDATE
        22 10/06/2004 22:02:48  WEEKNIGHT_WINDOW  OPEN
        25 10/07/2004 06:59:37  WEEKNIGHT_WINDOW  CLOSE
        26 10/07/2004 22:01:37  WEEKNIGHT_WINDOW  OPEN
        29 10/08/2004 06:59:51  WEEKNIGHT_WINDOW  CLOSE
```

The `DBA_SCHEDULER_WINDOWS_DETAILS` view provides information about every window that was active and is now closed (completed). The following statement shows sample output from that view:

```
SELECT LOG_ID, WINDOW_NAME, ACTUAL_START_DATE, ACTUAL_DURATION
  FROM DBA_SCHEDULER_WINDOW_DETAILS;

    LOG_ID WINDOW_NAME      ACTUAL_START_DATE                    ACTUAL_DURATION
---------- ---------------- ------------------------------------ ---------------
        25 WEEKNIGHT_WINDOW 06-OCT-04 10:02.48.832438 PM PST8PDT +000 01:02:32
        29 WEEKNIGHT_WINDOW 07-OCT-04 10.01.37.025704 PM PST8PDT +000 03:02:00
```

Notice that log IDs correspond in both of these views, and that in this case the rows in the DBA_SCHEDULER_WINDOWS_DETAILS view correspond to the CLOSE operations in the DBA_SCHEDULER_WINDOW_LOG view.

> **See Also:**
>
> - *Oracle Database Reference* for details on the *_SCHEDULER_WINDOW_LOG view
> - *Oracle Database Reference* for details on the DBA_SCHEDULER_WINDOWS_DETAILS view

29.2.3.3 Purging Logs

To prevent job and window logs from growing indiscriminately, use the SET_SCHEDULER_ATTRIBUTE procedure to specify how much history (in days) to keep.

Once per day, the Scheduler automatically purges all log entries that are older than the specified history period from both the job log and the window log. The default history period is 30 days. For example, to change the history period to 90 days, issue the following statement:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','90');
```

Some job classes are more important than others. Because of this, you can override this global history setting by using a class-specific setting. For example, suppose that there are three job classes (class1, class2, and class3), and that you want to keep 10 days of history for the window log, class1, and class3, but 30 days for class2. To achieve this, issue the following statements:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','10');
DBMS_SCHEDULER.SET_ATTRIBUTE('class2','log_history','30');
```

You can also set the class-specific history when creating the job class.

Note that log entries pertaining to steps of a chain run are not purged until the entries for the main chain job are purged.

Purging Logs Manually

The PURGE_LOG procedure enables you to manually purge logs. As an example, the following statement purges all entries from both the job and window logs:

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

Another example is the following, which purges all entries from the jog log that are older than three days. The window log is not affected by this statement.

```
DBMS_SCHEDULER.PURGE_LOG(log_history => 3, which_log => 'JOB_LOG');
```

The following statement purges all window log entries older than 10 days and all job log entries older than 10 days that relate to job1 and to the jobs in class2:

```
DBMS_SCHEDULER.PURGE_LOG(log_history => 10, job_name => 'job1, sys.class2');
```

29.2.4 DBMS_SCHEDULER In-Memory Trace

The DBMS_SCHEDULER In-Memory tracing feature provides a tool for temporarily storing the scheduler trace messages generated during process execution.

Due to the concurrent nature of the scheduler, several processes might be executed at same time, therefore allocation and usage of memory must be implemented efficiently to minimize the memory and disk consumption. In-memory tracing follows a circular form, that is the most recent trace entries overwrite the oldest ones and automatic dumps to disk will happen when the process hits a severe problem. This minimizes the amount of memory used to store trace messages while maximizing the amount of information collected. In-memory tracing eases collection of trace messages generated since the very first failure, reduces user interaction to collect traces, and avoids multiple requests for problem reproduction.

As memory used to store scheduler traces might be limited, the database administrator can define the amount of space reserved for tracing or even disable the tracing feature using the set of parameters shown below.

Table 29-3 Administration of Scheduler In-Memory Trace Features

SQL Statement	Example	Description
`alter system set "_scheduler_ora_buffer_size"=<SIZE>;`	Setting buffer size for Oracle Server processes to 32KB `alter system set "_scheduler_ora_buffer_size"=32;`	This SQL statement defines the size of the buffer storing trace from Oracle Server processes. It receives an integer value that defines the number of kilobytes (KB) of memory allocated by the process as an input. Typically, these processes perform top level calls that are used to create or alter scheduler objects. As thousands of server processes can be active at any time, the DBA must limit the maximum amount of buffer size per session (32KB per session) or limit the number of concurrent sessions in memory trace. Valid values are 0 to 1024. The default value is 32KB. Setting this value to 0 disables in-memory tracing for Oracle Server processes.
`alter system set "_scheduler_cjq0_buffer_size"=<SIZE>;`	Setting buffer size for CJQ process to 256 KB `alter system set "_scheduler_cjq0_buffer_size" = 256;`	This SQL statement defines the size of the buffer storing trace messages from scheduler coordinator process (CJQ0). It receives an integer value defining the number of KB of memory allocated by the process as an input. Valid values are 0 to 131072. The default value is 1024. Setting this value to 0 disables in-memory tracing for CJQ0 process.

Table 29-3 (Cont.) Administration of Scheduler In-Memory Trace Features

SQL Statement	Example	Description
`alter system set "_scheduler_jnnn_buffer_size"=<SIZE>;`	Setting buffer size for Jnnn process to 0 `alter system set "_scheduler_jnnn_buffer_size" = 0;`	This SQL statement defines the size of the buffer storing trace from scheduler slave processes. It receives an integer value defining the number of KB of memory allocated by the process as an input. Valid values are 0 to 1024. The default value is 32KB. Setting this value to 0 disables in-memory tracing for scheduler job slave processes.
`alter system set "_dump_scheduler_inmemory_trace_on_timeout"=<TRUE|FALSE>;`	`alter system set "_dump_scheduler_inmemory_trace_on_timeout"=false;`	This SQL statement enables or disables the automatic dump of trace when the completion time of some internal routines exceeds the value defined with `_scheduler_inmemory_trace_timeout`. If this parameter is set to TRUE, then the content of the memory buffer is dumped to persistent storage. Setting this parameter to FALSE will disable the automatic dumps. The default value is TRUE.
`alter system set "_dump_scheduler_inmemory_trace_on_error"=<TRUE|FALSE>;`	`alter system set "_dump_scheduler_inmemory_trace_on_error"=true;`	This SQL statement enables or disables the automatic dump of traces to persistent storage when a critical error is captured during some internal routines execution. If the parameter is set to TRUE, internal errors (for example, ORA-27352) will dump the current contents of memory trace of this process. Valid values are TRUE|FALSE. The default value is TRUE.
`alter system set "_scheduler_inmemory_trace_timeout"=<number_of_seconds>;`	`alter system set "_scheduler_inmemory_trace_timeout" = 300;`	This SQL statement defines the threshold to declare a call for an unexpected behavior. Following are the examples of an unexpected behavior: • A job query refresh taking more than 10 minutes to complete. • The time required to get a lock on scheduler object, for instance on a job queue, is taking more than one minute. Values are measured in seconds. Valid values are 60 to 1800. The default value is 600. Setting this value too low can generate excessive dumps.

29.2.5 Managing Scheduler Security

You should grant the appropriate privileges to users based on the Scheduler operations they will perform.

You should grant the `CREATE JOB` system privilege to regular users who need to be able to use the Scheduler to schedule and run jobs. You should grant `MANAGE SCHEDULER` to any database administrator who needs to manage system resources. Grant any other Scheduler system privilege or role with great caution. In particular, the `CREATE ANY JOB` system privilege and the `SCHEDULER_ADMIN` role, which includes it, are very powerful because they allow execution of code as any user. They should only be granted to very powerful roles or users.

Handling external job is a particularly important issue from a security point of view. Only users that need to run jobs outside of the database should be granted the `CREATE EXTERNAL JOB` system privilege that allows them to do so. Security for the Scheduler has no other special requirements. See *Oracle Database Security Guide* for details regarding security.

If users need to create credentials to authenticate their jobs to the operating system or a remote database, grant them `CREATE CREDENTIAL` system privilege.

> **Note:**
>
> When upgrading from Oracle Database 10*g* Release 1 (10.1) to Oracle Database 10*g* Release 2 (10.2) or later, `CREATE EXTERNAL JOB` is automatically granted to all users and roles that have the `CREATE JOB` privilege. Oracle recommends that you revoke this privilege from users that do not need it.

29.3 Import/Export and the Scheduler

You must use the Data Pump utilities (`impdp` and `expdp`) to export Scheduler objects.

You cannot use the earlier import utility (`IMP`) with the Scheduler. Also, Scheduler objects cannot be exported while the database is in read-only mode.

An export generates the DDL that was used to create the Scheduler objects. All attributes are exported. When an import is done, all the database objects are re-created in the new database. All schedules are stored with their time zones, which are maintained in the new database. For example, schedule "Monday at 1 PM PST in a database in San Francisco" would be the same if it was exported and imported to a database in Germany.

Although Scheduler credentials are exported, for security reasons, the passwords in these credentials are not exported. After you import Scheduler credentials, you must reset the passwords using the `SET_ATTRIBUTE` procedure of the `DBMS_SCHEDULER` package.

If the schema being exported has any programs with arguments, then the schema must be granted `CREATE TABLE` privilege beforehand. Otherwise, you will receive `ORA-01031` during the schema export.

Related Topics

- Oracle Data Pump

29.4 Troubleshooting the Scheduler

You can troubleshoot problems with Scheduler.

- A Job Does Not Run
  A job may fail to run for several reasons.
- A Program Becomes Disabled
  A program can become disabled if a program argument is dropped or `number_of_arguments` is changed so that all arguments are no longer defined.
- A Window Fails to Take Effect
  A window can fail to take effect for various reasons.

29.4.1 A Job Does Not Run

A job may fail to run for several reasons.

To begin troubleshooting a job that you suspect did not run, check the job state by issuing the following statement:

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS;
```

Typical output will resemble the following:

```
JOB_NAME                       STATE
------------------------------ ---------
MY_EMP_JOB                     DISABLED
MY_EMP_JOB1                    FAILED
MY_NEW_JOB1                    DISABLED
MY_NEW_JOB2                    BROKEN
MY_NEW_JOB3                    COMPLETED
```

- About Job States
  If a job does not run, then it can be in one of the following states: failed, broken, disabled, or completed.
- Viewing the Job Log
  The job log is an important troubleshooting tool.
- Troubleshooting Remote Jobs
  Remote jobs must successfully communicate with a Scheduler agent on the remote host. If a remote job does not run, then check the `DBA_SCHEDULER_JOBS` view and the job log first.
- About Job Recovery After a Failure
  The Scheduler can attempt to recover jobs that are interrupted.

29.4.1.1 About Job States

If a job does not run, then it can be in one of the following states: failed, broken, disabled, or completed.

- Failed Jobs
  If a job has the status of `FAILED` in the job table, then it was scheduled to run once but the execution has failed. If the job was specified as restartable, then all retries have failed.
- Broken Jobs
  A broken job is one that has exceeded a certain number of failures. This number is set in `max_failures`, and can be altered.

- Disabled Jobs
  A job can become disabled for several reasons.
- Completed Jobs
  A job will be completed if end_date or max_runs is reached.

29.4.1.1.1 Failed Jobs

If a job has the status of FAILED in the job table, then it was scheduled to run once but the execution has failed. If the job was specified as restartable, then all retries have failed.

If a job fails in the middle of execution, only the last transaction of that job is rolled back. If your job executes multiple transactions, then you must be careful about setting restartable to TRUE. You can query failed jobs by querying the *_SCHEDULER_JOB_RUN_DETAILS views.

29.4.1.1.2 Broken Jobs

A broken job is one that has exceeded a certain number of failures. This number is set in max_failures, and can be altered.

In the case of a broken job, the entire job is broken, and it will not be run until it has been fixed. For debugging and testing, you can use the RUN_JOB procedure.

You can query broken jobs by querying the *_SCHEDULER_JOBS and *_SCHEDULER_JOB_LOG views.

29.4.1.1.3 Disabled Jobs

A job can become disabled for several reasons.

The reasons include the following:

- The job was manually disabled
- The job class it belongs to was dropped
- The program, chain, or schedule that it points to was dropped
- A window or window group is its schedule and the window or window group is dropped

29.4.1.1.4 Completed Jobs

A job will be completed if end_date or max_runs is reached.

If a job recently completed successfully but is scheduled to run again, then the job state is SCHEDULED.

29.4.1.2 Viewing the Job Log

The job log is an important troubleshooting tool.

For details and instructions, see "Viewing the Job Log".

29.4.1.3 Troubleshooting Remote Jobs

Remote jobs must successfully communicate with a Scheduler agent on the remote host. If a remote job does not run, then check the DBA_SCHEDULER_JOBS view and the job log first.

Then perform the following tasks:

1. Check that the remote system is reachable over the network with tools such as `nslookup` and `ping`.
2. Check the status of the Scheduler agent on the remote host by calling the `GET_AGENT_VERSION` package procedure.

   ```
   DECLARE
     versionnum VARCHAR2(30);
   BEGIN
     versionnum := DBMS_SCHEDULER.GET_AGENT_VERSION('remote_host.example.com');
     DBMS_OUTPUT.PUT_LINE(versionnum);
   END;
   /
   ```

   If an error is generated, the agent may not be installed or may not be registered with your local database. See "Using the Oracle Scheduler Agent to Run Remote Jobs" for instructions for installing, registering, and starting the Scheduler agent.

29.4.1.4 About Job Recovery After a Failure

The Scheduler can attempt to recover jobs that are interrupted.

The Scheduler attempts to recover jobs that are interrupted when:

- The database abnormally shuts down
- A job child process is terminated or otherwise fails
- For an external job, the external job process that starts the executable or script is terminated or otherwise fails. (The external job process is `extjob` on UNIX. On Windows, it is the external job service.)
- For an external job, the process that runs the end-user executable or script is terminated or otherwise fails.

Job recovery proceeds as follows:

- The Scheduler adds an entry to the job log for the instance of the job that was running when the failure occurred. In the log entry, the `OPERATION` is '`RUN`', the `STATUS` is '`STOPPED`', and `ADDITIONAL_INFO` contains one of the following:
  - `REASON="Job slave process was terminated"`
  - `REASON="ORA-01014: ORACLE shutdown in progress"`
- If `restartable` is set to `TRUE` for the job, the job is restarted.
- If `restartable` is set to `FALSE` for the job:
  - If the job is a run-once job and `auto_drop` is set to `TRUE`, the job run is done and the job is dropped.
  - If the job is a run-once job and `auto_drop` is set to `FALSE`, the job is disabled and the job `state` is set to '`STOPPED`'.
  - If the job is a repeating job, the Scheduler schedules the next job run and the job `state` is set to '`SCHEDULED`'.

When a job is restarted as a result of this recovery process, the new run is entered into the job log with the operation '`RECOVERY_RUN`'.

29.4.2 A Program Becomes Disabled

A program can become disabled if a program argument is dropped or number_of_arguments is changed so that all arguments are no longer defined.

See "Creating and Managing Programs to Define Jobs" for more information regarding programs.

29.4.3 A Window Fails to Take Effect

A window can fail to take effect for various reasons.

A window can fail to take effect for the following reasons:

- A window becomes disabled when it is at the end of its schedule
- A window that points to a schedule that no longer exists is disabled

See "Managing Job Scheduling and Job Priorities with Windows" for more information regarding windows.

29.5 Examples of Using the Scheduler

Examples illustrate using Scheduler.

- Examples of Creating Job Classes
  Examples illustrate creating job classes.
- Examples of Setting Attributes
  Examples illustrate setting attributes.
- Examples of Creating Chains
  Examples illustrate creating chains.
- Examples of Creating Jobs and Schedules Based on Events
  Examples illustrate creating event-based jobs and event schedules.
- Example of Creating a Job In an Oracle Data Guard Environment
  In an Oracle Data Guard environment, the Scheduler includes additional support for two database roles: primary and logical standby. You can configure a job to run only when the database is in the primary role or only when the database is in the logical standby role.

29.5.1 Examples of Creating Job Classes

Examples illustrate creating job classes.

To create a job class, you use the CREATE_JOB_CLASS procedure.

Example 29-1 Creating a Job Class

The following statement creates a job class:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'my_class1',
   service                    =>  'my_service1',
   comments                   =>  'This is my first job class');
END;
/
```

This creates my_class1 in SYS. It uses a service called my_service1. To verify that the job class was created, issue the following statement:

```
SELECT JOB_CLASS_NAME FROM DBA_SCHEDULER_JOB_CLASSES
WHERE JOB_CLASS_NAME = 'MY_CLASS1';

JOB_CLASS_NAME
------------------------------
MY_CLASS1
```

Example 29-2 Creating a Job Class

The following statement creates a job class:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name            =>  'finance_jobs',
   resource_consumer_group   =>  'finance_group',
   service                   =>  'accounting',
   comments                  =>  'All finance jobs');
END;
/
```

This creates finance_jobs in SYS. It assigns a resource consumer group called finance_group, and designates service affinity for the accounting service. Note that if the accounting service is mapped to a resource consumer group other than finance_group, jobs in this class run under the finance_group consumer group, because the resource_consumer_group attribute takes precedence.

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the CREATE_JOB_CLASS procedure and "Creating Job Classes" for further information

29.5.2 Examples of Setting Attributes

Examples illustrate setting attributes.

To set attributes, you use SET_ATTRIBUTE and SET_SCHEDULER_ATTRIBUTE procedures.

Example 29-3 Setting the Repeat Interval Attribute

The following example resets the frequency that my_emp_job1 runs daily:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_emp_job1',
   attribute      =>   'repeat_interval',
   value          =>   'FREQ=DAILY');
END;
/
```

To verify the change, issue the following statement:

```
SELECT JOB_NAME, REPEAT_INTERVAL FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME =  'MY_EMP_JOB1';
```

```
JOB_NAME             REPEAT_INTERVAL
----------------     ---------------
MY_EMP_JOB1          FREQ=DAILY
```

Example 29-4   Setting Multiple Job Attributes for a Set of Jobs

The following example sets four different attributes for each of five jobs:

```
DECLARE
 newattr sys.jobattr;
 newattrarr sys.jobattr_array;
 j number;
BEGIN
 -- Create new JOBATTR array
 newattrarr := sys.jobattr_array();

 -- Allocate enough space in the array
 newattrarr.extend(20);
 j := 1;
 FOR i IN 1..5 LOOP
   -- Create and initialize a JOBATTR object type
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'MAX_FAILURES',
                          attr_value => 5);
   -- Add it to the array.
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'COMMENTS',
                          attr_value => 'Test job');
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'END_DATE',
                          attr_value => systimestamp + interval '24' hour);
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'SCHEDULE_LIMIT',
                          attr_value => interval '1' hour);
   newattrarr(j) := newattr;
   j := j + 1;
 END LOOP;

 -- Call SET_JOB_ATTRIBUTES to set all 20 set attributes in one transaction
 DBMS_SCHEDULER.SET_JOB_ATTRIBUTES(newattrarr, 'TRANSACTIONAL');
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `SET_SCHEDULER_ATTRIBUTE` procedure and "Setting Scheduler Preferences"

29.5.3 Examples of Creating Chains

Examples illustrate creating chains.

To create chains, you use the `CREATE_CHAIN` procedure. After creating a chain, you add steps to the chain with the `DEFINE_CHAIN_STEP` or `DEFINE_CHAIN_EVENT_STEP` procedures and define the rules with the `DEFINE_CHAIN_RULE` procedure.

Example 29-5   Creating a Chain

The following example creates a chain where `my_program1` runs before `my_program2` and `my_program3`. `my_program2` and `my_program3` run in parallel after `my_program1` has completed.

The user for this example must have the `CREATE EVALUATION CONTEXT`, `CREATE RULE`, and `CREATE RULE SET` privileges. See "Setting Chain Privileges" for more information.

```
BEGIN
  DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name            =>  'my_chain1',
   rule_set_name         =>  NULL,
   evaluation_interval   =>  NULL,
   comments              =>  NULL);
END;
/

--- define three steps for this chain. Referenced programs must be enabled.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepA', 'my_program1');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepB', 'my_program2');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepC', 'my_program3');
END;
/

--- define corresponding rules for the chain.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE('my_chain1', 'TRUE', 'START stepA');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain1', 'stepA COMPLETED', 'Start stepB, stepC');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain1', 'stepB COMPLETED AND stepC COMPLETED', 'END');
END;
/

--- enable the chain
BEGIN
 DBMS_SCHEDULER.ENABLE('my_chain1');
END;
/

--- create a chain job to start the chain daily at 1:00 p.m.
BEGIN
 DBMS_SCHEDULER.CREATE_JOB (
   job_name        => 'chain_job_1',
   job_type        => 'CHAIN',
   job_action      => 'my_chain1',
   repeat_interval => 'freq=daily;byhour=13;byminute=0;bysecond=0',
   enabled         => TRUE);
END;
/
```

Example 29-6   Creating a Chain

The following example creates a chain where first `my_program1` runs. If it succeeds, `my_program2` runs; otherwise, `my_program3` runs.

```
BEGIN
 DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name            => 'my_chain2',
   rule_set_name         => NULL,
   evaluation_interval   => NULL,
   comments              => NULL);
END;
/

--- define three steps for this chain.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step1', 'my_program1');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step2', 'my_program2');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step3', 'my_program3');
END;
/

--- define corresponding rules for the chain.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE ('my_chain2', 'TRUE', 'START step1');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step1 SUCCEEDED', 'Start step2');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step1 COMPLETED AND step1 NOT SUCCEEDED', 'Start step3');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step2 COMPLETED OR step3 COMPLETED', 'END');
END;
/
```

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `CREATE_CHAIN`, `DEFINE_CHAIN_STEP`, and `DEFINE_CHAIN_RULE` procedures and "Setting Scheduler Preferences"

29.5.4 Examples of Creating Jobs and Schedules Based on Events

Examples illustrate creating event-based jobs and event schedules.

To create event-based jobs, you use the `CREATE_JOB` procedure. To create event-based schedules, you use the `CREATE_EVENT_SCHEDULE` procedure.

These examples assume the existence of an application that, when it detects the arrival of a file on a system, enqueues an event onto the queue `my_events_q`.

Example 29-7   Creating an Event-Based Schedule

The following example illustrates creating a schedule that can be used to start a job whenever the Scheduler receives an event indicating that a file arrived on the system before 9AM:

```
BEGIN
  DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
   schedule_name     =>  'scott.file_arrival',
   start_date        =>  systimestamp,
   event_condition   =>  'tab.user_data.object_owner = ''SCOTT''
      and tab.user_data.event_name = ''FILE_ARRIVAL''
      and extract hour from tab.user_data.event_timestamp < 9',
   queue_spec        =>  'my_events_q');
END;
/
```

Example 29-8   Creating an Event-Based Job

The following example creates a job that starts when the Scheduler receives an event indicating that a file arrived on the system:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  my_job,
   program_name        =>  my_program,
   start_date          =>  '15-JUL-04 1.00.00AM US/Pacific',
   event_condition     =>  'tab.user_data.event_name = ''LOW_INVENTORY''',
   queue_spec          =>  'my_events_q'
   enabled             =>  TRUE,
   comments            =>  'my event-based job');
END;
/
```

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information about the `CREATE_JOB` and `CREATE_EVENT_SCHEDULE` procedures

29.5.5 Example of Creating a Job In an Oracle Data Guard Environment

In an Oracle Data Guard environment, the Scheduler includes additional support for two database roles: primary and logical standby. You can configure a job to run only when the database is in the primary role or only when the database is in the logical standby role.

To do so, you set the `database_role` attribute. This example explains how to enable a job to run in both database roles. The method used is to create two copies of the job and assign a different `database_role` attribute to each.

By default, a job runs when the database is in the role that it was in when the job was created. You can run the same job in both roles using the following steps:

1. Copy the job
2. Enable the new job
3. Change the `database_role` attribute of the new job to the required role

The example starts by creating a job called `primary_job` on the primary database. It then makes a copy of this job and sets its `database_role` attribute to '`LOGICAL STANDBY`'. If the primary database then becomes a logical standby, the job continues to run according to its schedule.

When you copy a job, the new job is disabled, so you must enable the new job.

```
BEGIN DBMS_SCHEDULER.CREATE_JOB (
     job_name       => 'primary_job',
     program_name   => 'my_prog',
     schedule_name  => 'my_sched');

 DBMS_SCHEDULER.COPY_JOB('primary_job','standby_job');
 DBMS_SCHEDULER.ENABLE(name=>'standby_job', commit_semantics=>'ABSORB_ERRORS');
 DBMS_SCHEDULER.SET_ATTRIBUTE('standby_job','database_role','LOGICAL STANDBY');
END;
/
```

After you execute this example, the data in the `DBA_SCHEDULER_JOB_ROLES` view is as follows:

```
SELECT JOB_NAME, DATABASE_ROLE FROM DBA_SCHEDULER_JOB_ROLES
   WHERE JOB_NAME IN ('PRIMARY_JOB','STANDBY_JOB');

JOB_NAME                DATABASE_ROLE
--------                ----------------
PRIMARY_JOB             PRIMARY
STABDBY_JOB             LOGICAL STANDBY
```

Note:

For a physical standby database, any changes made to Scheduler objects or any database changes made by Scheduler jobs on the primary database are applied to the physical standby like any other database changes.

29.6 Scheduler Reference

There are several privileges and data dictionary views related to Scheduler.

- Scheduler Privileges
  Users can be granted various Scheduler privileges.
- Scheduler Data Dictionary Views
  You can query a set of views for information about Scheduler.

29.6.1 Scheduler Privileges

Users can be granted various Scheduler privileges.

Table 29-4 and Table 29-5 describe the various Scheduler privileges.

Table 29-4 Scheduler System Privileges

Privilege Name	Operations Authorized
`CREATE JOB`	This privilege enables you to create jobs, chains, schedules, programs, file watchers, destinations, and groups in your own schema. You can always alter and drop these objects in your own schema, even if you do not have the `CREATE JOB` privilege. In this case, the object would have been created in your schema by another user with the `CREATE ANY JOB` privilege.

Table 29-4 (Cont.) Scheduler System Privileges

Privilege Name	Operations Authorized
CREATE ANY JOB	This privilege enables you to create, alter, and drop jobs, chains, schedules, programs, file watchers, destinations, and groups in any schema except SYS. This privilege is extremely powerful and should be used with care because it allows the grantee to execute any PL/SQL code as any other database user.
CREATE EXTERNAL JOB	This privilege is required to create jobs that run outside of the database. Owners of jobs of type 'EXECUTABLE' or jobs that point to programs of type 'EXECUTABLE' require this privilege. To run a job of type 'EXECUTABLE', you must have this privilege and the CREATE JOB privilege. This privilege is also required to retrieve files from a remote host and to save files to one or more remote hosts.
EXECUTE ANY PROGRAM	This privilege enables your jobs to use programs or chains from any schema.
EXECUTE ANY CLASS	This privilege enables your jobs to run under any job class.
MANAGE SCHEDULER	This is the most important privilege for administering the Scheduler. It enables you to create and drop job classes, windows, and window groups, and to stop jobs with the force option, but not alter them. It also enables you to set and retrieve Scheduler attributes, purge Scheduler logs, and set the agent password for a database. Only SYS users can alter these objects.

Table 29-5 Scheduler Object Privileges

Privilege Name	Operations Authorized
SELECT	You can grant object privileges on a group to other users by granting SELECT on the group.
EXECUTE	You can grant this privilege only on programs, chains, file watchers, credentials, and job classes. The EXECUTE privilege enables you to reference the object in a job. It also enables you to view the object if the object is was not created in your schema.
ALTER	This privilege enables you to alter or drop the object it is granted on. Altering includes such operations as enabling, disabling, defining or dropping program arguments, setting or resetting job argument values and running a job. Certain restricted attributes of jobs of job type EXECUTABLE cannot be altered using the ALTER object privilege. These include job_type, job_action, number_of_arguments, event_spec, and setting PL/SQL date functions as schedules. For programs, jobs, chains, file watchers, and credentials, this privilege also enables schemas that do not own these objects to view them. This privilege can be granted on jobs, chains, programs, schedules, file watchers, and credentials. For other types of Scheduler objects, you must grant the MANAGE SCHEDULER system privilege.
ALL	This privilege authorizes operations allowed by all other object privileges possible for a given object. It can be granted on jobs, programs, chains, schedules, file watchers, credentials, and job classes.

Note:

No object privileges are required to use a destination object created by another user.

The SCHEDULER_ADMIN role is created with all of the system privileges shown in Table 29-4 (with the ADMIN option). The SCHEDULER_ADMIN role is granted to DBA (with the ADMIN option).

When calling DBMS_SCHEDULER procedures and functions from a definer's rights PL/SQL block, object privileges must be granted directly to the calling user. As with all PL/SQL stored procedures, DBMS_SCHEDULER ignores privileges granted through roles on database objects when called from a definer's rights PL/SQL block.

The following object privileges are granted to PUBLIC: SELECT ALL_SCHEDULER_* views, SELECT USER_SCHEDULER_* views, SELECT SYS.SCHEDULER$_JOBSUFFIX_S (for generating a job name), and EXECUTE SYS.DEFAULT_JOB_CLASS.

29.6.2 Scheduler Data Dictionary Views

You can query a set of views for information about Scheduler.

Some views are specific to multitenant container databases (CDBs), whereas others have a CDB-specific column. The V$ and GV$ views have a CON_ID column that identifies a container whose data is represented by a CDB_* row. CDB_* views correspond to all Scheduler DBA_* views. In a PDB, these views only show objects visible through a corresponding DBA_* view, but all objects are visible in the root. The CDB_* view contains all columns found in a given DBA_* view and the column (CON_ID).

Table 29-6 contains views associated with the Scheduler. The *_SCHEDULER_JOBS, *_SCHEDULER_SCHEDULES, *_SCHEDULER_PROGRAMS, *_SCHEDULER_RUNNING_JOBS, *_SCHEDULER_JOB_LOG, *_SCHEDULER_JOB_RUN_DETAILS views are particularly useful for managing jobs. See *Oracle Database Reference* for details regarding Scheduler views.

> **Note:**
>
> In the following table, the asterisk at the beginning of a view name can be replaced with DBA, ALL, or USER.

Example 29-9 Displaying Details About a Scheduler Job

This example shows information for completed instances of my_job1:

```
SELECT JOB_NAME, STATUS, ERROR#
FROM DBA_SCHEDULER_JOB_RUN_DETAILS WHERE JOB_NAME = 'MY_JOB1';

JOB_NAME     STATUS          ERROR#
--------     --------------  ------
MY_JOB1      FAILURE          20000
```

Table 29-6 Scheduler Views

View	Description
*_SCHEDULER_CHAIN_RULES	These views show all rules for all chains.
*_SCHEDULER_CHAIN_STEPS	These views show all steps for all chains.
*_SCHEDULER_CHAINS	These views show all chains.
*_SCHEDULER_CREDENTIALS *_CREDENTIALS	These views show all credentials. ** *_SCHEDULER_CREDENTIALS is deprecated in Oracle Database 12*c*, but remains available, for reasons of backward compatibility. The recommended view is *_CREDENTIALS.
*_SCHEDULER_DB_DESTS	These views show all database destinations.

Table 29-6 (Cont.) Scheduler Views

View	Description
*_SCHEDULER_DESTS	These views show all destinations, both database and external.
*_SCHEDULER_EXTERNAL_DESTS	These views show all external destinations.
*_SCHEDULER_FILE_WATCHERS	These views show all file watchers.
*_SCHEDULER_GLOBAL_ATTRIBUTE	These views show the current values of Scheduler attributes.
*_SCHEDULER_GROUP_MEMBERS	These views show all group members in all groups.
*_SCHEDULER_GROUPS	These views show all groups.
*_SCHEDULER_INCOMPATIBILITY	These views show all programs or jobs that are members of incompatibility definitions.
*_SCHEDULER_JOB_ARGS	These views show all set argument values for all jobs.
*_SCHEDULER_JOB_CLASSES	These views show all job classes.
*_SCHEDULER_JOB_DESTS	These views show the state of both local jobs and jobs at remote destinations, including child jobs of multiple-destination jobs. You obtain job destination IDs (job_dest_id) from these views.
*_SCHEDULER_JOB_LOG	These views show job runs and state changes, depending on the logging level set.
*_SCHEDULER_JOB_ROLES	These views show all jobs by Oracle Data Guard database role.
*_SCHEDULER_JOB_RUN_DETAILS	These views show all completed (failed or successful) job runs.
*_SCHEDULER_JOBS	These views show all jobs, enabled as well as disabled.
*_SCHEDULER_NOTIFICATIONS	These views show all job state e-mail notifications.
*_SCHEDULER_PROGRAM_ARGS	These views show all arguments defined for all programs as well as the default values if they exist.
*_SCHEDULER_PROGRAMS	These views show all programs.
*_SCHEDULER_REMOTE_DATABASES	These views show information about the remote databases accessible to the current user that have been registered as sources and destinations for remote database jobs.
*_SCHEDULER_REMOTE_JOBSTATE	These views displays information about the state of the jobs accessible to the current user at remote databases.
*_SCHEDULER_RESOURCES	These views describe the resource metadata.
*_SCHEDULER_RUNNING_CHAINS	These views show all chains that are running.
*_SCHEDULER_RUNNING_JOBS	These views show state information on all jobs that are currently being run.
*_SCHEDULER_RSRC_CONSTRAINTS	These views show the types of resources used by a job or program and the number of units of each resource it needs.
*_SCHEDULER_SCHEDULES	These views show all schedules.
*_SCHEDULER_WINDOW_DETAILS	These views show all completed window runs.
*_SCHEDULER_WINDOW_GROUPS	These views show all window groups.
*_SCHEDULER_WINDOW_LOG	These views show all state changes made to windows.
*_SCHEDULER_WINDOWS	These views show all windows.
*_SCHEDULER_WINGROUP_MEMBERS	These views show the members of all window groups, one row for each group member.