# Documentum Job Scheduling and agent_exec

**Author:** Thomas Harris
**Date:** 11/30/2007

# Table of Contents

## Introduction

The dm_agent_exec utility, “dm_agent_exec.exe” on Windows or “dm_agent_exec” on Solaris/Linux/UNIX, is a stand-alone program responsible for running scheduled jobs on the content server.

Agent_exec can be invoked in two different modes, either as the “method launcher” that runs a specific job, or the “agent_exec utility” which is responsible for creating new method launchers to execute jobs that are due to be run. On startup, agent_exec decides which mode it will run in based on the command line arguments that are passed to it.

## Program Arguments

| | |
|---|---|
| -docbase_name | The name of the server that the agent_exec utility will be associated with, in *docbase_name.server_config_name* format. |
| -docbase_owner | The name of the user that owns the docbase, which is used to connect to the content server. |
| -sleep_duration | The amount of time in seconds that the agent_exec utility sleeps in between checking for new jobs to execute. When not specified, the sleep duration defaults to 60 seconds. |
| -trace_level | A number indicating the level of tracing for the agent_exec process. 0 indicates no tracing, and 1 or greater enables all trace statements. |
| -job_id | The id of the dm_job object that a method launcher instance uses to determine which job to run.<br><br>This argument controls whether or not the agent_exec process will be the agent_exec utility or a method launcher. When the argument is supplied, the process will be a method launcher. |
| -override_sleep_duration | Overrides the argument specified in sleep_duration. |
| -max_concurrent_jobs | The maximum number of jobs that an agent_exec utility will allow to execute at the same time. |

**Warning:** You should never use the docbase_name, docbase_owner, job_id, or sleep_duration in the method_verb arguments. These are supplied by the content server.

See the “agent_exec Utility” section for information on changing these arguments.

## Tracing

To turn on tracing, use Documentum Administrator to add the -trace_level argument to the agent_exec_method method command line. For example:

.\dm_agent_exec -trace_level 1

Setting the trace level to any 1 will turn on full tracing for the process.
The log file is named agentexec.log and is stored in the $DOCUMENTUM/dba/log/*repository_id*/agentexec directory.

There is tracing for both the agent_exec utility and the method launchers. All trace statements from the agent_exec utility go into “agentexec.log”, while trace statements from method launchers go into separate files named “job_[job_id]”.

## Lock Files

Lock files are created both by the agent_exec utility and method launcher processes. All lock files go to the same directory as the agent_exec log files, which is: $DOCUMENTUM/dba/log/*repository_id*/agentexec

The lock file for the agent_exec utility is called agent_exec.lck.

The lock files for method launchers are called job*_[job_id]*.lck.

The purpose of the agent_exec.lck file is to make sure that only one agent_exec utility is running per server. If you attempt to start a second agent_exec utility process on the same server, it will fail due to the existence of the lock file.

The purpose of the job lock files is to allow the agent_exec utility to determine if a method launcher is still running.

## agent_exec Utility

In Documentum the agent_exec utility is responsible for deciding which jobs will be run, when they will be run, and in what order. It also sets the future execution times for jobs.

During startup the content server will attempt to launch the agent_exec utility. This agent_exec utility will be the only instance of agent_exec allowed to launch jobs set to run on the associated content server.

The agent_exec process is defined in the docbase as a dm_method object with the name “agent_exec_method”.

To retrieve this method in IAPI:

```
API> retrieve,c,dm_method where object_name = 'agent_exec_method'
```

Once you have retrieved the method object, you can add additional command line arguments by modifying the method_verb property.

**Warning:** You should never add custom values for the docbase_name, docbase_owner, job_id, or sleep_duration in the method_verb arguments as they are added automatically by the content server when it launches agent_exec. If you want to change the sleep duration, use override_sleep_duration rather than sleep_duration.

For example, to change max concurrent jobs to 4:

```
API> set,c,l,method_verb
SET> .\dm_agent_exec.exe -max_concurrent_jobs 4
```

After launching the agent_exec utility, the content server will periodically check to see if it is still running. If it has crashed, content server will attempt to launch it again once every minute.

Problems with startup of the agent_exec utility will cause the following errors to appear in the content server log:

- `[DM_SESSION_W_RESTART_AGENT_EXEC]warning:  "The agent exec program has stopped running.  It will be restarted."`
    - Occurs if the content server started the agent_exec process, but 1 minute later it is no longer running. The most likely reason for this is that agent_exec failed to connect to the docbase.
- `[DM_SESSION_W_AGENT_EXEC_FAILURE_EXCEED]warning:  "The failure limit of the agent exec program has exceeded. It will not be restarted again. Please correct the problem and restart the server."`
    - Occurs after the first error message has been shown 4 consecutive times.
    - The content server will not automatically attempt to launch agent_exec again unless the content server itself is restarted.

After going through the startup process, the agent_exec utility enters an infinite loop where it executes two tasks, then sleeps for the amount of time specified in sleep_duration or override_sleep_duration. By default this is 60 seconds. The first task is garbage collection of jobs that were launched on the current server and did not successfully finish. The second is the creation of method launchers to run jobs that are scheduled to be started.

*Garbage Collection*
The agent_exec utility queries the content server to see if there are any jobs that were once running on the same machine and are no longer running, but failed to finish successfully.

It first queries the content server to obtain a list of any dm_job objects that have a last invocation date set but no completion date, or have been checked out by an agent_exec process. It then checks to see if those jobs should be running on the current machine by

checking the target_server property of the object. For purposes of garbage collection, it considers any job belonging to the same target docbase and machine as belonging to it. (The target_server attribute has a format of *docbase_name[.config_name]@host_name*)

If it determines that the job was in fact launched on the current machine, it then checks to see if the file lock created by the method launcher still exists. This file lock is created when the method launcher process starts and removed automatically when it ends. If the lock is found, then the process is still running and it will nothing will be done. If the lock is not found, then the job must have been launched on the current server but is no longer running, which indicates that execution failed.

If a job meets all of these conditions, the properties of the job are reset so that it can be run again.

When a job has its properties cleaned up, an error message is generated stating that a "dead job" was found on the server:

- `[ERROR] [AGENTEXEC 5604] Detected while processing dead job` *`job_name`*`: The job object indicated the job was in progress, but the job was not actually running.  It is likely that the dm_agent_exec utility was stopped while the job was in progress.`

Sometimes "processing dead job" error messages will contain a DMCL error or some other error message that indicates some problem while trying to modify the job's properties.

An isolated occurrence of this error message with no additional errors does not necessarily indicate a serious problem. (It may just be that the content server shut down before the job finished, etc.)

*Creating Method Launchers*

After the garbage collection task has been completed, the agent_exec utility queries the content server to retrieve a list of jobs that are currently are scheduled to run. After a list of these jobs is retrieved, it iterates through each job and attempts to launch it.

The agent_exec utility then verifies the properties of the job to see if it can be launched. It is at this time that the target server and other properties are checked. (This is discussed in more detail in the troubleshooting section.)

A lock file for the current job is then created in the same directory as the agent_exec log files. This is used by the agent_exec utility in the garbage collection procedure to determine if a method launcher it created is still running.

Various properties of the job, most importantly the next invocation time, are set at this point.

**Note:** The next invocation time for a job is set to be the current value of a_next_invocation plus the time interval. The exception is if that time has already passed. In that case it will instead be set to the closest time in the future that makes sense. So if a job set to run at 5:00 PM on a 10 minute interval is started at 5:11 PM, it will be set to run next at 5:20 PM, and so forth.

A method launcher is then started asynchronously, and agent_exec will sleep for 30 seconds. It then moves on to the next job.

If there are no more jobs, or if the maximum number of jobs is already executing, as set by the max_concurrent_jobs argument, then agent_exec will sleep for the amount of time specified in the sleep_duration argument. When it wakes up, it will run the garbage collection procedure and look for jobs to run again.

## Method Launcher

The method launcher is a simple program that runs the job specified in the job_id command line argument.

The method launcher procedure is as follows:

- Connect to the docbase.
- Modify the attributes of the job object to indicate the process id of the method launcher, the current status of the job (that it is executing), and the time the job was started. Also set the completion date to nulldate.
- Invoke the method on the server.
- Update the job object with the results of the method, the time the job was completed, etc.
- Update other job object attributes so that the job can be run again.
- Disconnect from the docbase.

## Job Scheduling Troubleshooting

*Determining Eligible Jobs*
The agent_exec utility uses a DQL query to determine which jobs are eligible to be run and in what order it will attempt to run them.

Use this query to see which jobs are eligible to run right now:

```
SELECT ALL r_object_id, object_name, a_next_invocation FROM
dm_job WHERE ( (run_now = 1) OR ((is_inactive = 0) AND ( (
a_next_invocation <= DATE('now') AND a_next_invocation IS NOT
NULLDATE ) OR ( a_next_continuation <= DATE('now') AND
a_next_continuation IS NOT NULLDATE ) ) AND ((expiration_date >
```

```
DATE('now')) OR (expiration_date IS NULLDATE)) AND
((max_iterations = 0) OR (a_iterations < max_iterations))) ) AND
(i_is_reference = 0 OR i_is_reference is NULL) AND (i_is_replica
= 0 OR i_is_replica is NULL) ORDER BY a_next_invocation,
r_object_id
```

Use this query to see which jobs are eligible to be run right now, as well as what jobs are eligible to be run at a future date:

```
SELECT ALL r_object_id, object_name, a_next_invocation FROM
dm_job WHERE ( (run_now = 1) OR ((is_inactive = 0) AND ( (
a_next_invocation IS NOT NULLDATE ) OR (  a_next_continuation IS
NOT NULLDATE ) ) AND ((expiration_date > DATE('now')) OR
(expiration_date IS NULLDATE)) AND ((max_iterations = 0) OR
(a_iterations < max_iterations))) ) AND (i_is_reference = 0 OR
i_is_reference is NULL) AND (i_is_replica = 0 OR i_is_replica is
NULL) ORDER BY a_next_invocation, r_object_id
```

This is the first query, only without the constraint that either the next invocation or continuation time has already passed.

If a job shows up in the first query, it is eligible to run right now and agent_exec will pick it up when it uses an almost identical query to check for eligible jobs in the future. If it shows up in the second, then it will be eligible to run at some point in the future when a_next_invocation or a_next_continuation is greater than the current time.

If a job does not appear in the second query, then it will never run at any point in the future. The job object's properties must be modified in some way in order to make it run.

*Additional Requirements*
Even if a job appears in the queries above, then it is not guaranteed to execute once the next invocation time or next continuation time passes. There are other requirements that must be met for the job to be executed.

Among the reasons why the agent_exec utility may not actually execute a job:

- agent_exec may determine that the job's properties are "invalid". If this is an issue, then you should see a fairly descriptive error message that will explain what the problem is.

  For reference, a list of reasons why a job can be considered invalid:

  - The job may have no method name specified.
  - The start date of the job is set to null or is not formed correctly.
  - The start date of the job is before the year 1970.
  - The next invocation date of the job is null or is not formed correctly.
  - The next invocation date of the job is before the year 1970.
  - The expiration date of the job is not formed correctly.

- The expiration date of the job is before the year 1970 or is later than the start date.
- The max iterations of the job is set to a number less than 0.
- The run_mode and run_interval values for the job object conflict. For example, if run_mode is set to be a specific day of the month (run_mode = 8) and run_interval is set to 33.

**Note:** If a job is found to be invalid for one of the reasons above, agent_exec may set the is_inactive attribute on the job object to "T" after it occurs, which will prevent the job from being found by query again. If this is the case, the properties of the job must be fixed and is_inactive must be set back to “F” so it will run again.

- The target server may be set to a different server. Note that this is not specified in the original query. Unless the docbase name, server config name, and host name match what is stored in the target_server property, this check will fail.
- A DMCL error or some other error can occur in agent_exec sometime after the query is run but before the job is launched. Should this happen an error message should appear in the log showing the failed DMCL command and an e-mail to the administrator should be sent as well.
- The owner of the job (specified in the owner_name of a dm_job object) does not have superuser privileges. No error message will be generated and the job will never execute if this is the case.
- The job object has already been checked out by anyone, even the same user that is running agent_exec. It must be checked in before it can be processed by agent_exec.
- An agent_exec method launcher process that was created to launch the same job at an earlier date is still running on the same machine. This is determined by seeing whether or not the file lock for the job still exists on the system.

*Differences between Invocation Time and When Jobs Actually Start*
Certain aspects job scheduling system may prevent jobs from being started at the expected time.

Some reasons that this can occur:

- The agent_exec utility always sleeps for 30 seconds after launching a new job. There is no way to change this behavior. This means that there is always a gap of at least 30 seconds in between jobs being launched.
- Additional jobs will not be launched if a certain number of jobs are already running. By default, only 3 jobs can be executed at the same time. This setting can be overridden with the -max_concurrent_jobs command line argument.
- Jobs that are retrieved in the first query above are ordered by their next invocation time, with the earliest time being first. This is the order that agent_exec will attempt to execute the jobs.

Thus it can take longer than expected to clear a backlog of jobs if there are either a great number of jobs that need to be executed (because of the waiting period in between launching new jobs means a new job can be started at most every 30 seconds) or a few jobs that take a long time to execute (since agent_exec will not allow other jobs to execute if too many are running).

Also note that if many jobs are all set to run at the same time on the same interval they are unlikely to run exactly when expected because of the 30 second gap in between launching jobs and possibly because of the limit on concurrent jobs. For example, if 10 jobs have a next invocation time of 3PM and a run interval of 5 minutes, the earliest that the 10th job can be launched is 3:05 PM.

Also note that if any of these jobs is launched after 3:05 PM, then it will have its next invocation time set to 3:10 PM when it is launched and will have only run once in that 10 minute period rather than twice as would be expected.