Job Control Guide

Schrödinger Suite 2012 Update 2

Schrödinger Press

Revision A, September 2012

Contents

Document Conventions

In addition to the use of italics for names of documents, the font conventions that are used in this document are summarized in the table below.

Font	Example	Use
Sans serif	Project Table	Names of GUI features, such as panels, menus, menu items, buttons, and labels
Monospace	`$SCHRODINGER/maestro`	File names, directory names, commands, environment variables, command input and output
Italic	*filename*	Text that the user must replace with a value
Sans serif uppercase	CTRL+H	Keyboard keys

Links to other locations in the current document or to other PDF documents are colored like this: Document Conventions.

In descriptions of command syntax, the following UNIX conventions are used: braces { } enclose a choice of required items, square brackets [ ] enclose optional items, and the bar symbol | separates items in a list from which one item must be chosen. Lines of command syntax that wrap should be interpreted as a single command.

File name, path, and environment variable syntax is generally given with the UNIX conventions. To obtain the Windows conventions, replace the forward slash / with the backslash \ in path or directory names, and replace the $ at the beginning of an environment variable with a % at each end. For example, `$SCHRODINGER/maestro` becomes `%SCHRODINGER%\maestro`.

Keyboard references are given in the Windows convention by default, with Mac equivalents in parentheses, for example CTRL+H (⌘H). Where Mac equivalents are not given, COMMAND should be read in place of CTRL. The convention CTRL-H is not used.

In this document, to *type* text means to type the required text in the specified location, and to *enter* text means to type the required text, then press the ENTER key.

References to literature sources are given in square brackets, like this: [10].

Introduction

Nearly all computational jobs launched from Maestro are run under Schrödinger's Job Control facility. The Job Control facility provides a uniform mechanism for launching, monitoring and controlling calculations, both for jobs launched from Maestro and for jobs launched from the command line. The Job Control facility keeps information on jobs in a database that is set up for each user.

The provision of a Job Control facility makes it easy to run Schrödinger software. You can submit jobs *from* any computer *to* any computer or cluster, or to a batch queue, and have the results returned to the computer from which you submitted the job. The process is independent of the particular platform from which you submit the job or on which you run the job. The platform details are hidden, but that also means that the details have to be communicated to Job Control.

The setup and configuration tasks necessary for Job Control to work are described in Chapter 7 of the *Installation Guide*. In this manual, Chapter 2 describes the life cycle of a job and provides information on submitting jobs and how the job execution is set up. Chapter 3 provides information on the job database and managing jobs.

Schrödinger software was originally designed to run on hosts that run some version of the UNIX operating system (including Linux and vendor-specific implementations of UNIX). On Linux, you can run jobs and Job Control commands from the command line. On Windows, many of the programs and commands can be run directly from a Windows shell. Two shells are provided with the installation that have the Schrödinger environment set up:

- Schrödinger Command Prompt—DOS shell.
- Schrödinger Power Shell—Windows Power Shell (if available).

You can open these shells from Start > All Programs > Schrodinger-2012. If you want access to Unix-style utilities (such as `awk`, `grep`, and `sed`) or commands that are not available in these shells, simply preface the commands with `sh`, or type `sh` in either of these shells to start a Unix-style shell.

1.1 Notes

- Job Control changes made for Suite 2011 are not compatible with earlier releases. You must ensure that the hosts file does not point to software installations for earlier releases, and that the location of the job database is different from that for earlier releases (this is the case by default, if you do not explicitly set the location of the job database).

Running Jobs

Jobs can be submitted to a designated host from a Maestro session (either UNIX or Windows) or from the UNIX command line. The details of setting up the data in Maestro and of submitting jobs from the command line for a given product are described in the documentation for the product. However, there are some common elements that relate to the job submission and execution, and the incorporation of results into a Maestro project. These common elements are described in two sections of this chapter.

When you run a Schrödinger job on a particular host, Job Control determines where to find the hosts file, which version of the software to use, which environment variables need to be passed to the host, where the scratch directory is located, how to make the input files available to the job, how to retrieve the output files, and how to incorporate them into a Maestro project. An overview of this process is provided in the first section of this chapter. The remaining sections provide details on how Job Control obtains information and sets up the runtime environment for the job.

2.1 The Job Life Cycle

To understand how information is obtained and passed on by Job Control, it is useful to have an understanding of how Schrödinger jobs are run. The "life cycle" of a job can be summarized as follows.

1. A top-level script is run locally (on the *submission host*). This script parses command-line arguments relating to Job Control, sets some environment variables, and locates a software installation that is compatible with the submission host and a software installation that is compatible with the host on which the program will actually be run (the *execution host*). This information is cached locally, and updated periodically.
2. The top-level script then runs the startup script for the program locally. This script parses the command arguments for the program, and assembles information on input and output files for the program.
3. The startup script then runs a job launch script locally, called `jlaunch`. This script assigns a job ID, creates a record in the job database, and populates it with initial values. It also verifies that the required input files exist.

4. If the job is to be run on a remote host, `jlaunch` starts a script locally, called `jserver`, that sets up a socket connection to the remote host that can be used for file transfer. For local jobs, `jserver` is also started, but it does not need to manage file transfer, only the interaction with the job database.
5. The job launch script then runs a script on the execution host to start the actual program. This script is called `jmonitor`. If the job is remote, it copies input files to the execution host using the connection established by `jserver`, runs the program, and copies output files back to the submission host. The program itself is responsible for checking in and checking out licenses.
6. The job launch script takes care of cleanup and incorporation into a Maestro project, if required, and updates the job record with the final status.

If you submit a job to a batch queue, there are some extra tasks that need to be performed. The life cycle of a batch job is as follows.

1. A top-level script is run locally (on the *submission host*). This script parses command-line arguments relating to Job Control, sets some environment variables, and locates a software installation on the submission host and a software installation on the host that manages the batch queue (the *queue manager*). This host could be one of a few designated nodes on a cluster, for example.
2. The top-level script then runs the startup script for the program locally. This script parses the command arguments for the program, and assembles information on input and output files for the program.
3. The startup script then runs a job launch script locally, called `jlaunch`. This script creates a record in the job database and populates it with initial values. It also creates a batch script for the job and submits this script to the queue.
4. The job launch script starts a script locally, called `jserver`, that sets up a connection to the queue manager that can be used for file transfer. By default, this connection is an SSH tunnel, but if secure transfer is not in effect, a socket connection is used. Another `jserver` process is started on the queue manager (or other designated host) that sets up connections to the compute nodes when the queued jobs are started. This process acts as a proxy for the file transfer, and is done because it is not always possible to connect directly from the submission host to the compute nodes. This instance of `jserver` is also known as a `jproxy` process.
5. When the queued job is started on an execution host (a cluster node, for example), it runs `jmonitor`. Before performing other tasks, `jmonitor` looks for a compatible software installation on the execution host. It then copies input files to the execution host, runs the program (which checks out and checks in licenses), and copies output files back to the

submission host. The copying is done via the SSH tunnel or the sockets opened by `jserver` processes on the manager node and the submission host. If the queue has license checking, the queueing software checks for license availability. See Section 7.4 of the *Installation Guide* for information on setting up license checking.

6. The job launch script takes care of cleanup and incorporation into a Maestro project, if required, and updates the job record with the final status.

Jobs that are distributed over multiple processors, usually have a script that manages the distribution of subjobs to the individual processors. Each subjob is then executed by `jmonitor`.

2.2 Running Jobs From Maestro

For most jobs, Maestro opens the Start dialog box so that you can make job settings and select options for handling of the job, including product-dependent options. The dialog box is usually named *task* – Start, where *task* is a name related to the product, job, or initiating panel. The controls available in the Start dialog box vary according to the job being run. (Some products and solutions have separate locations in which job settings are made.)

When you start Maestro, the settings in the `schrodinger.hosts` file are used to determine the available options in the Start dialog box and other job setting controls. Maestro searches the following directories for a `schrodinger.hosts` file, in the order given, and uses the first one that it finds.

- The Maestro startup directory, which is the directory in which you started Maestro (Linux) or the `Schrodinger` folder in your documents folder (Windows or Mac)
- The Schrödinger user resources directory, `$HOME/.schrodinger` (Linux or Mac) or `%USERPROFILE%\Schrodinger` (Windows)
- The installation directory

Under UNIX, you can always determine which `schrodinger.hosts` file is being used by entering the following command in the directory in which you started Maestro:

```
$SCHRODINGER/program -HOSTS
```

where *program* is any Schrödinger executable. The options in the Start dialog box are passed on to Job Control.

The Start dialog box can have two sections, Output and Job. The components available in each section are described below. Some of the components are present in all instances of this dialog box, others are only present in some instances. An example of a Start dialog box is shown in Figure 2.1.

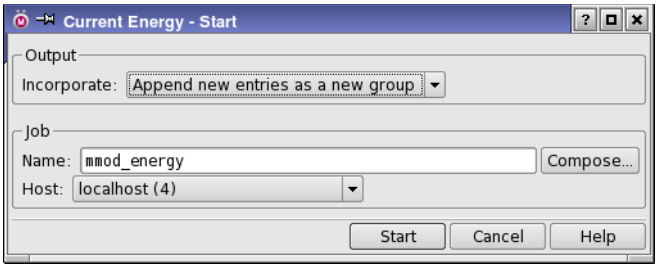

Figure 2.1. The Start dialog box.

Output section

- Incorporate option menu—Choose whether the new entries are appended to the project either individually or as an entry group, replace the existing entries, or are not incorporated into the project at all. See Section 2.9 on page 19 for more information on incorporating output.

  The choice that you make from this menu is persistent for a given job type: the next time you run a job of that type, the incorporation mode that you last used is the default mode. The incorporation mode is stored as a preference by Maestro, so the choice persists across Maestro sessions.

Job section

- Title text box—Enter a title for the input structure file. The default is the entry title.
- Name text box—Enter a name for the computational job. This name is used as a prefix for all files created during the calculation. A default name is assigned based on the job being run. Job names cannot contain spaces or nonprinting characters.

  Note: Maestro does *not* automatically assign new names to jobs or files. If files of the same name exist, a warning is displayed before any files are overwritten.

- Compose button—You can compose a job name by selecting from the existing job names in the project or the file names of files in the working directory, then editing the name. To do so, click Compose to open the Compose Job Name dialog box, select an item from one of the two lists, then edit it in the Job name text box. When you click OK, the job name is entered in the Name text box in the Start dialog box.
- Standard Names—Choose a job name from a list of standard names. The standard names are composed using values of job parameters and a serial number. This option menu is only available for products for which standard names are defined in the job name resource file.

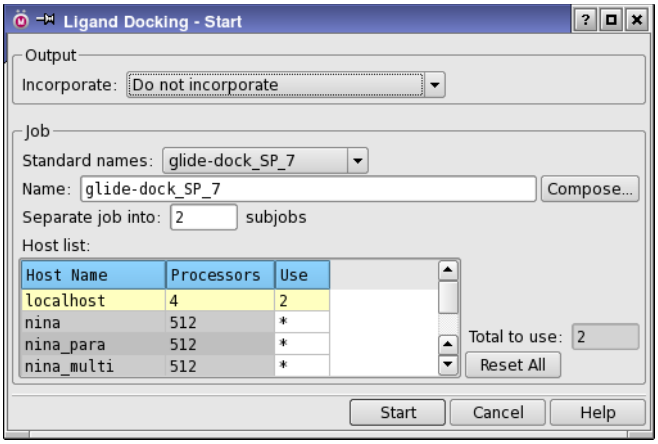

Figure 2.2. The Start dialog box for distributable jobs.

The job name resource file, `jobname.res`, is stored in `$SCHRODINGER/maestro-v`*version*`/data/res`. You can copy this file to your own resource directory (`$HOME/.schrodinger/maestro`*NN* on Linux or `%APPDATA%\Schrodinger\maestro`*NN* on Windows) or to the working directory and edit it to define your own standard names. The job serial number is stored in your resource directory as `jobserial.res`. Removing this file resets the serial number to 1.

- Host—Choose a host on which to run the job. This option menu displays all the hosts defined in the hosts file, with the number of processors on the host in parentheses.

  The host `localhost` means the host on which you are running Maestro. If you run a job locally, Maestro automatically reduces the priority of the job so that it does not compete with Maestro for resources. The exceptions are Hydrophobic/philic map jobs, and structure cleanup jobs. To change this behavior, set the `SCHRODINGER_NICE` environment variable (see Table B.1 on page 41).

- Username—Enter your user name, if it is required for running the job on remote hosts. The default value is the user name of the user who started Maestro, unless there is a `username` setting for this host in the hosts file or the `SCHRODINGER_REMOTE_USER` environment variable is set, for remote jobs. If the user name is not correct for the selected host, you can change it in this text box.

- CPUs—Specify the number of processors to use to run the job. The number of processors (or cores) actually used is limited by the number of licenses available for the type of job.

- Scratch directory—Choose a directory to store temporary files.

Once you have finished setting these options, you can click Start to start the job.

For jobs that can be distributed over multiple hosts, the Host option menu is replaced by the Host List table. This table displays all the hosts defined in the hosts file, with the number of processors available on the host in the Processors column. The Use column specifies the number of processors to use from the given host. The default number is the number available. You can edit this column to set the number of processors, and you can reset the values to the default by clicking Reset All.

To specify the hosts and number of processors to use for a job, edit the values in the Use column as needed, and select the desired hosts in the table. You can select multiple hosts with shift-click and control-click. Only the selected hosts will be used for the job. The total number of processors for the job is reported in the Total to use text box.

For some distributable jobs you can specify the number of subjobs to run independently from the number of processors. The number of subjobs should not be smaller than the number of processors, and in most cases should be several times the number of processors for good load balancing. To specify the number of subjobs, enter the desired value into the Separate job into *N* subjobs text box. See *Running Distributed Schrödinger Jobs* for more information.

2.3 Running Jobs From the Command Line

For most purposes, you can start jobs from Maestro. You can also run or submit jobs from the command line on Linux hosts or from a Schrödinger shell on Windows hosts (see below). The Job Control facility recognizes a number of command-line options that can be used to control the behavior of the job. These options are summarized in Table 2.1.

In addition to these options, the startup scripts for some programs support several other options. These options are summarized in Table 2.2. You should check which options are supported by entering the command

```
$SCHRODINGER/program -HELP
```

Table 2.1. Command options.

Option	Description
`-DISP` *policy*	Set the project incorporation policy for this job: `ignore`, `append`, `appendungrouped`, `workspace`, or `replace`. Requires `-PROJ`, and overrides any user setting of `SCHRODINGER_JOB_DISPOSITION`. The policies are described in Section 2.9 on page 19. Default: `ignore`.
`-DRIVERHOST` *host*	For distributed jobs, specify the host on which to run the driver ("master job"). Must specify a single host. If omitted, the host on which the driver runs depends on the application. See

Table 2.1. Command options. (Continued)

Option	Description
`-HOST` *host* `-HOST` *host*`:`*n* `-HOST "`*host1*`:`*n1 host2 ...*`"`	Run a job on the specified host or submit a job to the specified batch queue. *host* is the value of a `name` entry in the hosts file or the actual address of a host. To specify multiple hosts, supply a blank-separated list in quotes. To specify multiple processors on a host, append a colon and the number of processors to the host name. For distributed jobs, if `-DRIVERHOST` is used, `-HOST` specifies the subjob hosts; if `-SUBHOST` is used, `-HOST` specifies the driver host. `-HOST` is ignored if both `-DRIVERHOST` and `-SUBHOST` are used. Default: run on the local host.
`-NICE`	Run the job at reduced priority. On UNIX, the program is run with `nice -19`. On Windows, the program is run at "idle" priority. Overrides any user setting of `SCHRODINGER_NICE`.
`-NOLAUNCH`	Perform all the steps necessary for launching the job, but stop short of actually launching it.
`-NONICE`	Do not run the job at reduced priority. Overrides any user setting of `SCHRODINGER_NICE`.
`-NOSTARTUP`	In the process of launching the job, stop short of executing the application startup script. Mainly useful for debugging.
`-PROJ` *projectname*	Assign the job to a Maestro project. Overrides any user setting of `SCHRODINGER_PROJECT`
`-QARGS` *queue-args*	Pass arguments to the queue manager. These arguments are appended to those specified by the `qargs` settings in the hosts file and the `SCHRODINGER_QUEUE_ARGS` environment variable.
`-SAVE`	Copy the archived contents of the job directory back to the submission directory after the job finishes. The name of the archive is *jobid*`-jobdir.zip`.
`-SUBHOST` *host* `-SUBHOST` *host*`:`*n* `-SUBHOST "`*host1*`:`*n1 host2 ...*`"`	For distributed jobs, specify the hosts on which the subjobs will run. The syntax is the same as for `-HOST`. Default: run subjobs on the hosts specified by `-HOST`.
`-TMPDIR` *directory*	Specify the scratch directory for the job. The job directory is created as a subdirectory of the scratch directory. Overrides any user setting of `SCHRODINGER_TMPDIR`
`-USER` *username*	Specify the user name to be used for remote jobs. Must be used with `-HOST`. Default: use the value of `SCHRODINGER_REMOTE_USER`, if set, otherwise use the same user name as on the submission host.

Table 2.2. Options supported by some programs

Option	Description
`-INTERVAL` *time*	Interval in seconds between updates of the monitored files (usually the log file). The files are copied back to the submission directory at the specified time interval.
`-LOCAL`	Write temporary files in the submission directory instead of the scratch directory. Input and output files are not copied. Default: write temporary files in the scratch directory.
`-NOJOBID`	Run the job outside of Job Control. If you use this option, you are responsible for managing all the environment variables, file handling, and checking done by Job Control.
`-NOLOCAL`	Write temporary files in the scratch directory instead of the submission directory. Some programs and utilities write files locally when run on the local host; this option can be used to force the use of the scratch directory.
`-WAIT`	Wait for the job to finish before executing another command. In a terminal window, this means that the command prompt is not displayed until the job finishes. In a script, it means that the next command is not executed until the job finishes. Default: return control to the shell immediately.

Table 2.3. Information options.

Option	Description
`-DEBUG` `-DDEBUG`	Show the details of operation of the top-level script. Show the verbose details of operation of the top-level script.
`-ENTRY`	Show the section of the `schrodinger.hosts` file that will be used for this job.
`-HELP`	Display command syntax for the application.
`-HOSTS`	List the hosts that are available for calculations.
`-LIST` [*version-options*]	List the available versions of the program that can be run on the host specified by `-HOST`. If no host is specified, the local host is used. If v*ersion-options* is `-ALL`, list all available versions, even if not compatible with the specified host.
`-WHICH` [*version-options*]	Show which version of the program and of the `mmshare` library would be used for the given version options.
`-WHY` [*version-options*]	Gives information about why the specified version was selected.

Command-line options always take precedence over the corresponding environment variable. Some of the options from Table 2.1 and Table 2.2 are described in more detail below. You can also obtain information for each program about the hosts you can use, and which version of the program will be run. These options are listed in Table 2.3.

The allowed values of *version-options* for the information options are listed in Table 2.4. These options can also be used to select the version of the software that will be run. For example, to run Jaguar 7.5, you could enter the command

```
$SCHRODINGER/jaguar -REL v7.5 jobname.in
```

Table 2.4. Version options.

Option	Description
`-ARCH` *platform*	Select the version for the given platform, e.g., `Linux-x86`, `IRIX-mips4`.
`-COMPAT` *executable*	Select the version that is compatible with the specified executable, for which the full path must be given.
`-REL` *version*	Release version number: `v42`, `v4.2`, `42`, `v42062`, `42062`, `v4.2.062` are all acceptable forms.
`-VER` *pattern*	If you have multiple versions installed, you can specify a pattern with this option that matches the installation path to use for your job. The default installation is the one printed by `-WHICH` with no version arguments.

On Windows, you can run Unix (Linux) commands by opening a Schrödinger Command Prompt window from the Start menu, then entering the `sh` command. The `unxutils` package provides many of the commands available in a Unix shell.

2.3.1 Running Scripts

You can run scripts from the command line with the following command:

```
$SCHRODINGER/run scriptname [-FROM product] [-FIND|-FINDALL] [options]
```

This command runs a top-level script like those for Schrödinger products, which recognizes the Job Control options listed in the tables above (except for Table 2.2) and sets up environment variables. The script name can be one of the scripts supplied with the distribution or downloaded from the Script Center, or it can be one of your own scripts. The script will not run under Job Control unless it has been set up to do so. Any options that are not recognized by the top-level script are passed on to your script.

The run command looks for scripts in the following locations in the order given:

1. The current directory.
2. The directory defined by the MMSHARE_EXEC environment variable, which is set by the run command to $SCHRODINGER/mmshare-v*version*/bin/*platform*, where version is the mmshare version. (You should not set this environment variable yourself.)
3. The directory defined by the SCHRODINGER_SCRIPTS environment variable.
4. The scripts*m.n* directory in your Schrödinger user resource area, which is $HOME/.schrodinger on Linux and %APPDATA%\Schrodinger on Windows, and *m.n* is the mmshare 2-digit version. This is the directory used for command-line scripts that are downloaded for personal use from the web site using the Update Scripts From Website panel in Maestro.
5. The directory $SCHRODINGER/mmshare-v*version*/python/common.
6. The directory $SCHRODINGER/mmshare-v*version*/python/scripts.
7. The directory $SCHRODINGER/mmshare-v*version*/python.
8. The directory defined by the MAESTRO_SCRIPT_LOCATION environment variable.
9. The maestro*vv*/scripts directory in your Schrödinger user resource area, where *vv* is the 2-digit Maestro version number. This is the directory used for Maestro scripts that are downloaded for personal use from the web site using the Update Scripts From Website panel in Maestro.
10. Your execution path, defined by the PATH environment variable.

The -FIND option displays the path used to find the script, telling you which of these directories would be used to run the script. The -FINDALL option displays the path to all of the possible locations where the script could be found, in the order given in the list above. These options are useful when you have several versions of a script and want to find out which one is being used, or which versions you have available.

If you use the -FROM *product* option, the executable directory for the product that you specify, $SCHRODINGER/*product*-v*version*/bin/*platform*, replaces MMSHARE_EXEC in the hierarchy.

The run command is especially useful for scripts that make use of the Schrödinger libraries or Python modules, for which the environment is set up when you use this command.

2.3.2 The HOST, DRIVERHOST, and SUBHOST Options

Jobs can be submitted to a remote host using the `-HOST` option to specify the remote host name. This name must be the value of a `name` setting in the hosts file (a "host entry name"), or an actual host name. For example:

`$SCHRODINGER/bmin -HOST` *host jobname*

For programs that can run a single job in parallel or distribute several jobs over a number of processors, the `-HOST` option can be used to specify the list of hosts to be used. The host list is a list of host entry names, separated by spaces. The list must be enclosed in quotes if there is more than one host specified.

Each host entry name can also specify a processor count, using the syntax:

host:*processors*

An example of a host list specification is:

```
-HOST "florence:2 glinda"
```

The first host in the list is the main host for the job, that is, the host on which the driver process for the parallel or distributed job runs. The entire set of hosts is used for the subjobs, including the first.

For many products, the host on which the driver runs and the host on which the subjobs run can be specified independently, with the `-DRIVERHOST` option and the `-SUBHOST` option. This is useful when you want the driver to run locally, for example. You might want to do this when running the subjobs on a cluster via a queueing system, so that the driver is not occupying a node on the cluster. This is not necessary for Phase, which runs a subjob on the same host as the driver, and ignores the driver host setting.

If you use either of these two options, the `-HOST` option serves as a fallback for the other option. So if you use `-SUBHOST`, `-HOST` only specifies the location of the driver; if you use `-DRIVERHOST`, `-HOST` only specifies the location of the subjobs. If you use both, `-HOST` is ignored.

The relationship between these options can be understood in terms of how the driver host and the subjob hosts are set. `-HOST` sets the driver host and the subjob hosts. `-DRIVERHOST` unconditionally sets the driver host (overriding `-HOST`), and `-SUBHOST` unconditionally sets the subjob hosts (overriding `-HOST`). The default for anything that is not set is `localhost`. This means that if you only set `-DRIVERHOST`, the entire job runs on the driver host.

Not all jobs use all of these settings. See *Running Distributed Schrödinger Jobs* for detailed information.

If you specify a host name that is not a host entry name, the settings for `localhost` in the hosts file are used for the job. You must ensure that there is a software installation on the specified host.

2.3.3 The WAIT option

All jobs are run in the background automatically, dissociated from the terminal session or application from which they were launched. As a result, the job continues to run even if you quit Maestro or the terminal session from which you launched the job. For command-line jobs, this means that the UNIX command prompt is displayed immediately, without waiting for the job to finish. This behavior is not always desirable, especially if you want to run the job in a script, in which some subsequent action can be taken only after the job finishes. The command-line option `-WAIT` can be used to prevent the shell from continuing to the next command until after the job finishes. For example:

```
$SCHRODINGER/bmin -WAIT job_name
```

Even with this option, however, the calculation is placed in the background, so pressing CTRL+C or CTRL+Z does not affect it. This option applies to jobs submitted to a batch queue as well as jobs that are run directly.

When the job finishes, the return code for the job is set to the return code for the computational program. Thus, if the program failed, the return code for the job indicates why it failed. The return code is also available in the ExitCode field of the job record.

2.3.4 The LOCAL Option

For some applications, you can request temporary files to be written to the submission directory (the directory from which you submitted the job), instead of to the job directory, which is a subdirectory of the scratch directory. This is done with the `-LOCAL` option. For example,

```
$SCHRODINGER/bmin -LOCAL jobname
```

Not every application supports this option. When it is supported, it may be used for both local and remote jobs. If you run a remote job with the `-LOCAL` option, it is important to make sure that the submission directory is accessible on the remote machine. This option also suppresses the copying of input and output files.

For some utilities, temporary files are written in the submission directory if the job is run locally. For these utilities, you can force the files to be written in the scratch directory with the `-NOLOCAL` option.

2.3.5 Location of the Hosts File

When you start a job from the command line, one of the first tasks of Job Control is to locate a hosts file. The hosts file used for a given job is the first one found in the following list:

- The file specified by the environment variable `SCHRODINGER_HOSTS`
- The `schrodinger.hosts` file in the current directory
- The `schrodinger.hosts` file in `$HOME/.schrodinger` (Linux) or `%USERPROFILE%\Schrodinger` (Windows)
- The `schrodinger.hosts` file in the Schrödinger software installation

For MacroModel jobs the following locations are searched before the list given above:

1. A file specified by the command line argument `-HOSTFILE`.
2. The file *jobname*`.hst` in the startup directory on the submission host, where *jobname* is the stem of the command file name for the current calculation (e.g., if the command file were called `cal_en1.com`, this file would be called `cal_en1.hst`).

The information in the hosts file is then used to make default settings for the job, such as the location of the scratch directory.

If the hosts file is specified by the environment variable `SCHRODINGER_HOSTS`, this file is also used to locate the hosts for any subjobs.

2.4 Location of the Scratch Directory

Most jobs now run in a scratch directory by default, rather than in the directory from which the job was started (the *submission* directory, also called the *launch* directory). When a job runs in a scratch directory, a subdirectory is created in it for the job, named *tmpdir*/*username*/*unique_name*. Here, *tmpdir* is the path to the scratch directory. The job name is usually used for *unique_name*, but it can also have a sequence number or the job ID appended to it to make the directory name unique. This subdirectory is called the *job directory*. Input files are copied to the job directory, temporary files, log files, and output files are created in the job directory, and the output and log files are copied back to the submission directory when the job finishes.

If you submit a job from Maestro, you may be able to choose the scratch directory in the Start dialog box. Maestro reads all the `tmpdir` settings from the hosts file, and presents these in the Scratch directory option menu.

For jobs submitted from the command line, there are a number of ways to specify the *tmpdir* directory. Job Control uses the first specification found from the following list:

- The directory given on the command line with the `-TMPDIR` option. For example,

  ```
  $SCHRODINGER/bmin -TMPDIR /scr/mmod_tmp job_name
  ```

- The directory specified by the `SCHRODINGER_TMPDIR` environment variable, if this is set on the submission host.
- The directory specified by the first `tmpdir` setting for the host entry in the hosts file.

For jobs run on a remote host, the following locations are considered after the ones listed above if a scratch directory is not defined:

- The directory specified by the environment variable `SCHRODINGER_TMPDIR` on the remote host.
- The directory specified by the environment variable `TMPDIR` on the remote host.

If no other specification for *tmpdir* is found, the directory `$HOME/.schrodinger/tmp` is used. In this case, the *username* is not used to form the job directory name, since it would be redundant. Temporary files can occupy a large amount of disk space, so you are strongly advised to ensure that a suitable directory is defined on every host, preferably in the hosts file.

In all cases, *tmpdir* is created if it does not exist. If the file system on which *tmpdir* is to be created does not exist, the job fails.

When the job finishes, the job directory is automatically removed, if the following conditions are met:

- The output files were all successfully copied back to the submission directory.
- The directory did not exist before the job started.
- The `-SAVE` option was not used in the job submission.

Some programs allow you to force the storage of temporary files in the submission directory, by using the `-LOCAL` option to the command for running the program—see Section 2.3.4 on page 14.

2.5 Software Version Selection

When a job is started, the Job Control facility searches for compatible software installations that are available on the execution host. Compatibility here means that the platform (e.g. `Linux-x86`) matches the execution host; in other words, the execution host has access to a software version that it can run. The first location on the following list that contains a compatible software installation on the execution host is used:

1. Software in the directory specified by the SCHRODINGER environment variable on the submission host.

   This environment variable must be defined to submit a job. The entries in the hosts file are not used if this directory exists on the execution host and has a compatible software installation.

2. Software in the directory specified by the schrodinger settings in the entry for the host and the localhost entry in the hosts file.

   The hosts file used is the one on the submission host. These schrodinger settings are compiled into a list, which can be displayed on Linux by entering the command for the program with the -LIST option in a terminal window, for example:

   ```
   $SCHRODINGER/program -LIST
   ```

If there is more than one compatible installation in the selected location, the most recent is used unless options are set to define the software version to use.

Thus, all the information on the software locations must be available on the submission host. You can specify different installations for a given host entry by including multiple schrodinger definitions in the hosts file on the submission host. See Section 7.1 of the *Installation Guide* for information on setting up the hosts file.

The results of the remote commands used to locate a compatible software version are cached, to speed up the launching of remote jobs. If a newer version of a product is added to an installation (in an update release to a suite, or a patch, for example), it is possible that this caching mechanism could cause the older version of the product to continue to be used for up to a day after the new version was installed. You can clear the cache by using the following command on the launch host.

```
$SCHRODINGER/jobcontrol -resetcache
```

You can also turn off the caching mechanism altogether by setting the environment variable SCHRODINGER_NO_HUNT_CACHE to any non-empty value besides 0, and you can set the cache expiry period with the environment variable SCHRODINGER_HUNT_CACHE_EXPIRY.

If you submit a job to a batch queue, an additional check is done when the job starts using the information obtained on the submission host, which is passed on by Job Control. This is done because Job Control can only check the availability of software on the queue manager when the job is launched.

When you run a job from the command line, no checking is done to ensure compatibility of software versions between the submission host and the execution host unless requested. You can select a version, a release, and an architecture by specifying command-line options, as

described in Table 2.4 on page 11. Compatibility checking is requested automatically when Maestro submits a job, to ensure that the software that is used for the job is compatible with the Maestro version.

As software releases must be installed into separate directories, the selection of the software installation is essentially a selection of the software version. It is therefore possible that an older version of the software could be used to run a job: for example, if `$SCHRODINGER` is used to define the software location and it points to an older version, the execution host will run the software in the location specified by `$SCHRODINGER`, even if there is a newer version in another location. You should therefore ensure that your software installations are up-to-date, and that `$SCHRODINGER` is set to the version you want to use.

2.6 License File Location

When an executable starts or the queueing software checks the availability of licenses, the following locations are searched, in order, for the license file; the first one found is used.

1. `$SCHROD_LICENSE_FILE`
2. `$LM_LICENSE_FILE`
3. `$SCHRODINGER/license` or `$SCHRODINGER/license.txt`.

2.7 Environment Variables

Environment variables that can be used on the execution host can come from the following sources:

- Environment variables set in the shell startup script (`.bashrc`, `.cshrc`, and so on)
- Environment variables set on the submission host when the job is launched.
- Environment variables set in the hosts file for the execution host or `localhost`.

The environment variables are passed to the job in the following order of precedence:

1. Settings in the host entry.
2. Settings in the user environment on the submission host.
3. Settings in the user environment on the execution host. These are used as a fallback if those environment variables were not defined and exported with the job.

2.8 Input and Output Files

When you run a job from the command line you can specify input files in either of two ways: using absolute paths or relative paths. The way Job Control treats these differs. The configuration considerations for output files and for updates of the Job Control database are similar to those for input files specified using absolute paths. The basic rules are as follows:

- **Files specified using absolute paths.** These are paths that begin with a forward slash (/) such as `/home/unohu/prime/1mcp.mae`. Job Control first looks for the file on the execution host exactly as given. For instance, it would look for a file named `/home/unohu/prime/1mcp.mae` on the execution host. If it finds the file, it does nothing more, and the file on the execution host is used for input. If it does not find the file, it looks for a file of that name on the submission host, and copies it to the job directory of the execution host.

  If you specify an absolute path for a file on a remote host, you should check that this is the file you intended to use. There is no guarantee that the file that appears at the specified location on the execution host is the same as or has the same contents as one that appears at the specified location on the submission host; nor is there a requirement that a file of this name be accessible on the submission host.

  Specifying file names with absolute paths is useful if you have large files that you can copy to the desired location before submitting the job. For example, you might want to run several Glide docking jobs with the same grid. You could copy the grid files to the desired location, then specify them with an absolute path. Copying large files prior to job submission can help reduce network traffic, especially on clusters.

- **Files specified using relative paths with ../**—These files are copied from their locations on the submission host to the job directory on the execution host. For example, the file `../prime/1mcp.mae` is copied to *jobdir*`/1mcp.mae`.

- **Files in the submission directory or its subdirectories**—These files are copied with their relative path to the job directory on the execution host. Any subdirectories are created in the job directory. For example, `1mcp.mae` is copied to *jobdir*`/1mcp.mae`, and `prime/1mcp.mae` is copied to *jobdir*`/prime/1mcp.mae`.

Output files are copied back from the job directory to the submission directory.

2.9 Incorporation of Job Output

When Maestro project entries are used as input for jobs, the structure and property output can be incorporated into the project. The output files are always copied back to the submission directory: the incorporation options affect what is done in the Maestro project. You can specify how you want the results to be incorporated in the Start dialog box (see Section 2.2 on page 5).

To open the Start dialog box, click the Start button in the panel used to set up the job. The incorporation options and their effects are described below.

- Append new entries as a new group—New entries are added to the end of the entry list in the project as an entry group. The group name is the name of the file from which the entries originated, minus the extension. If there is only one entry, it is appended as an individual entry, not as a group. This is the default for jobs submitted from Maestro. Corresponds to the command-line option `-DISP append`.
- Append new entries individually—New entries are added to the end of the entry list in the project as individual entries. Corresponds to the command-line option `-DISP appendungrouped`.
- Replace existing entries—Incoming entries replace the existing entries from which they originated. Matching of the incoming and original entries is done by entry ID. Corresponds to the command-line option `-DISP replace`.
- Replace Workspace—The Workspace is replaced with the first structure from the output file, but no project entries are created or modified. Corresponds to the command-line option `-DISP workspace`.
- Do not incorporate—Structure and property data are not incorporated into the project. You can import the results later using the Import panel. See Chapter 3 of the *Maestro User Manual* for more information on importing structures. Corresponds to the command-line option `-DISP ignore`.

You can also specify a project and incorporation mode when you run jobs from the command line, by using the `-PROJ` and `-DISP` options—see Table 2.1 on page 8. The default for `-DISP` is `ignore` for jobs run from the command line, since the default for `-PROJ` is not to specify a project. The `ignore` option for `-DISP` is the same as the Do not incorporate option in Maestro.

By default, incorporation of job output into the project takes place only from monitoring mode in Maestro. If you are monitoring a job launched from the current project, and that job finishes during the monitoring session, incorporation is immediate. If you are not in monitoring mode when the job finishes, you must monitor the job in the Monitor panel to incorporate the results. This applies also to jobs run from the command line and associated with a project: you must monitor the job in the Monitor panel for incorporation to take place.

The default incorporation behavior can be changed in the Jobs tab of the Preferences panel (see Section 13.2.12 of the *Maestro User Manual*). You can choose to have jobs incorporate automatically, in which case jobs launched from the current project are incorporated as soon as they finish and are ready for incorporation. This means that you will not be able to use Maestro until the job has finished incorporating. You can also choose to receive a prompt for incorporation when a job is ready. If you chose not to incorporate the job at that time, you must monitor the job to incorporate it, or use the Incorporate All button in the Monitor panel.

When a job is incorporated, by default the first structure in the job output is displayed in the Workspace. For entries that are incorporated as an entry group, the entry list is scrolled to the first of these entries. You can change this behavior to minimize disruption to your work by setting a preference in the Jobs tab of the Preferences panel.

Incorporation can be undone. Undoing incorporation removes the new entries from the project and restores the Workspace to the state it was in just before monitoring mode was entered.

Computational programs normally propagate entry names (as well as other properties) from their input into their output structures. Where the relationship is one-to-many, as in a conformational search, consecutive structures in the output file have identical entry names. If the job is run with the "append" incorporation mode, the names are identical to that of the input entry. If the entries do not have a name, they are assigned a name that is constructed from the job name, with a "dot suffix." Thus, the output of a job named `ligand1` would be named `ligand1.1`, `ligand1.2`, and so on. Note that the entry name is no longer used as a unique identifier in the project: the entry ID is used instead.

Managing Jobs

The Job Control facility provides tools for monitoring and controlling the jobs that it runs. These tools have both a graphical user interface in the Maestro Monitor panel and a command-line interface in the `jobcontrol` command. The interface gets information from a job database that is set up for each user, and uses this information to perform various tasks, like providing job status and killing jobs.

You can monitor jobs that are run locally, run remotely, or run in a batch queue. Jobs submitted to a grid computing manager cannot be monitored, because Job Control does not have information from the grid manager. A script has been provided to obtain information on jobs running on a United Devices grid. Certain kinds of subjobs cannot be monitored: if they are launched from a master job that runs on a remote host, and the submission directory on the remote host is not mounted on the local host, the information is unavailable.

The first section in this chapter describes the job database. Subsequent sections describe the Monitor panel and the `jobcontrol` command. Use of the `jobcontrol` command is not supported on Windows hosts.

3.1 The Job Database

Information about each job is kept in the user's job database. The database contains a record for each job. You can determine which jobs are in the database by using the Maestro Monitor panel or the command-line `jobcontrol` utility.

By default, this database is kept in the directory `$HOME/.schrodinger/.jobdb2` on Linux and `%LOCALAPPDATA%\Schrodinger\.jobdb2` on Windows. If you want to change the location, you can set the environment variable `SCHRODINGER_JOBDB2` to the desired location. You should set this environment variable globally, in your `.bashrc` or `.cshrc` file on Linux, and in your system properties on Windows, so that you are always using the same job database. If you set this environment variable locally, Job Control will not be able to locate your jobs and they might not finish. (See Appendix A of the *Installation Guide* for information on setting environment variables on Windows.)

You must ensure that this database can be read and written by a job running on any host, either by making the directory directly available on the host, or by ensuring that the host has access to a host on which it is available by passwordless `ssh` (or `rsh`). If a job does not have access to the database, it will not be updated.

Note: The job database format changed in Suite 2011, and is not compatible with previous releases. You must *not* set the job database directory to a location used for releases prior to Suite 2011.

3.1.1 The Job Record

Each job has a job record in the job database. The job record is a list of fields, one on each line, each consisting of a field name and its value. Many of these fields contain information that is only useful to the job control system, but a number may also be useful to users. Some of the latter are listed in Table 3.1

Table 3.1. Fields used in the job record.

Field Name	Meaning
BackendPid	The PID for the program carrying out the calculation
Command	The command used to start up the actual calculation
Dir	The submission and output directory (on Host)
Envs	Environment variable settings for the job
Errors	Error messages from the job control system
ExitCode	Exit code for the job. Usually reflects the exit code for the computational program. See Table 3.2.
ExitStatus	The reason the calculation stopped
Host	The machine from which job was launched
InputFiles	Files copied from the submission directory to the job directory at startup
JobDir	The directory in which the job is run
JobHost	The machine on which the job is run
JobId	The job's JobId
JobPid	The PID for the job's `jmonitor` process, or the job's `jlaunch` process before `jmonitor` has started.
JobUser	The user name under which the job is run
LaunchTime	The time at which the job was submitted
LogFiles	Monitoring files that grow throughout the job
MonitorFiles	Files copied from the job directory to the output directory during monitoring
MonitorInterval	The interval in seconds between monitoring updates (0 if off)

Table 3.1. Fields used in the job record. (Continued)

Field Name	Meaning
Name	The job name
OutputFiles	Files copied from the job directory to the output directory at exit
ParentJobId	The JobId of the parent job, if this is a subjob
Processors	The number of processors used for a parallel or distributed job
Program	The program name
Project	The project name
StartTime	The time at which the calculation started running
Status	The current job status
StatusTime	The time at which Status was last updated
StopTime	The time at which the calculation stopped
StructureMonitorFile	The name of the monitoring file holding the molecular structure
StructureOutputFile	The name of the file holding the final molecular structure
Summary	Description of current job status
User	The user name under which job was launched
Warnings	Warning messages from the job or job control

To list the complete database record for a job, enter the command:

```
$SCHRODINGER/jobcontrol -dump jobid
```

Common values of ExitCode for the job are given in Table 3.2. On Linux, exit codes with a value greater than 128 are calculated by adding 128 to the signal number generated by the executable: thus exit code 137 corresponds to signal 9 (the kill signal).

3.1.2 Job Status

Every job proceeds through a series of well-defined stages (see Section 2.1 on page 3). The current stage of a job is displayed in the Status column of the Monitor panel. The Status of a completed job indicates the conditions under which it stopped. The job status descriptors that can appear in the Monitor panel are listed in Table 3.3. When program execution finishes, an exit status is returned, and is displayed with the completed status and the incorporated status in the Monitor panel. The exit status descriptors are listed in Table 3.4.

Table 3.2. Common job exit codes

Exit code	Meaning
0	Success
1	Failure, unspecified
15	Retriable licensing failure, such as inability to contact server
16	All licenses are in use
17	Fatal licensing failure, such as license does not exist
136	Floating-point error
137	Kill signal received
139	Segmentation fault
143	Termination signal received

Table 3.3. Job status descriptors in the Monitor *panel.*

Status	Meaning
launched	The job was submitted and assigned a JobId.
submitted	The job is in a batch queue, waiting to be scheduled.
started	The environment for the job is being set up.
running	The program is running.
paused	The program has temporarily been suspended.
exited	The program has stopped and the job is being cleaned up.
completed	The job has been cleaned up. This descriptor is followed by a colon and the exit status (see Table 3.4).
incorporated	The job results have been incorporated into a Maestro project.
stranded	Job Control could not retrieve results or clean up the job.

Table 3.4. Exit status descriptors in the Monitor *panel.*

Status	Meaning
finished	The program finished successfully.
stopped	The program was stopped at an appropriate point at the user's request.
killed	The job was killed by someone (not necessarily the user).
died	The program failed during execution.
fizzled	The job failed before the program could be run.

3.2 Managing Jobs From Maestro

You can manage jobs from the Monitor panel or from the command line. The common tasks can be performed from the Monitor panel, but the full range of job control tasks is only available from the command line. A web interface is also available for monitoring job progress, which can be started with `jobcontrol -g`.

To open the Monitor panel, choose Applications → Monitor Jobs, or click the Jobs button on the status bar in the main window. The Monitor panel opens automatically when a job starts if you have the preference set under Jobs–Monitoring in the Preferences panel.

The Monitor panel provides an interface to the job database. The panel has a jobs table and two tabs: Details, and File. Below the table is a row of action buttons. The table and the tabs are described in the following subsections.

When a job is started, by default Maestro immediately goes into monitoring mode. If the preference is set, the Monitor panel opens, and the log file for the job is displayed in the text area of the File tab. The display is updated as the log file changes. For some jobs, such as MacroModel and Jaguar geometry optimizations, the Workspace is updated with each new geometry as it is generated. You can choose whether to enter monitoring mode, whether to display the Monitor panel, and set other job-related values in the Preferences panel—see Section 13.2.12 of the *Maestro User Manual*.

Some panels, such as the Prime – Structure Prediction and Phase – Develop Pharmacophore Model panels, have a button that indicates whether a job is running. The button turns green and rotates when the job is running, and turns gray and stops turning when the job has finished. For these panels, the Monitor panel can be opened by clicking the button.

Most Maestro operations stop the monitoring of a job. You can resume or begin monitoring a job at any time by selecting it from the list of jobs in the Monitor panel.

3.2.1 The Jobs Table

The Jobs table lists jobs started by the current user. The job information is taken from the job database for the user and is updated periodically. You can select multiple rows in the table with the usual shift-click and control-click actions. If you select multiple rows, the Details and File tabs and the Monitor button are unavailable. You can apply an action to the selected jobs with the other action buttons. You can sort the table rows by clicking the heading of the column whose values you want to sort by. Subjobs are sorted in chronological order by default.

The table cells have tool tips that display the full content of the cell, except for the Status column, where an informative message describing the meaning of the status is displayed.

All jobs started by a user can be monitored in the Monitor panel, regardless of whether they were initiated from Maestro or from the command line. The class of jobs displayed can be selected from the Show option menu: all jobs, active jobs, or jobs from the current project.

To monitor a job, select the table row and click Monitor, or double-click the table row. The table row is highlighted in blue, and the File tab is placed on top with the log file displayed.

The Monitor panel allows jobs to be paused, resumed, or terminated using the Pause, Resume, Stop, and Kill buttons. Clicking Kill terminates the selected computational job immediately. The Stop button only affects MacroModel and Jaguar jobs, and is ignored by other jobs. When a job is stopped, the program saves results from the current stage of execution and cleans up before exiting. Pause and Resume can be used for all jobs.

You can clean up the job database by clicking the Clean Up button. Jobs that are completed and incorporated, or finished if they are not incorporatable, are removed from the database.

If a job fails, a postmortem archive of the job can be created for sending to technical support, by using the Postmortem button. The Create Postmortem dialog box opens, in which you can choose whether to include the structures in the archive (Include structures), whether to automatically modify path names in the files so they are unrecognizable (Automatically obfuscate path names), and create the archive. For each selected job, an archive is created and written to the current working directory on Linux or the Desktop on Windows. The archive is created with the `postmortem` utility, which is described in Appendix A. If you are concerned about sending confidential information, you can use this utility directly to control the content of the archive. An information dialog box provides the names of the archive files, including the path.

The frequency with which information on the currently monitored job is updated can be controlled with the box labeled Update the monitored job's status every *N* sec. You can enter any number in the text box, or use the arrow buttons to increase or decrease the value.

If you want to update the status of all running jobs, click Refresh.

To incorporate all completed jobs for the current project, click Incorporate All. Jobs that were submitted with incorporation method set to Workspace are incorporated, but no structure is displayed in the Workspace.

3.2.2 The Details Tab

The Details tab gives details of the job that is selected in the Jobs tab. The job information is repeated in the Job Summary text area. The Files table lists all files associated with the job except for the structure files. When you select a file in the table, the file is displayed in the File tab. If you double-click a file, the File tab is displayed with the selected file. By default, the log file is selected (or the first log file if there is more than one). Gray rows in the table mark files that are not available for viewing. The job record for the job is the last file in the list. This file shows details of the progress of the job and is useful for diagnosing errors.

3.2.3 The File Tab

The File tab displays the file that is selected in the Details tab. By default, this file is the log file for the job (or the first log file if there is more than one). The display is updated at regular intervals. When new text is added, the display scrolls to show any new text if the display is already

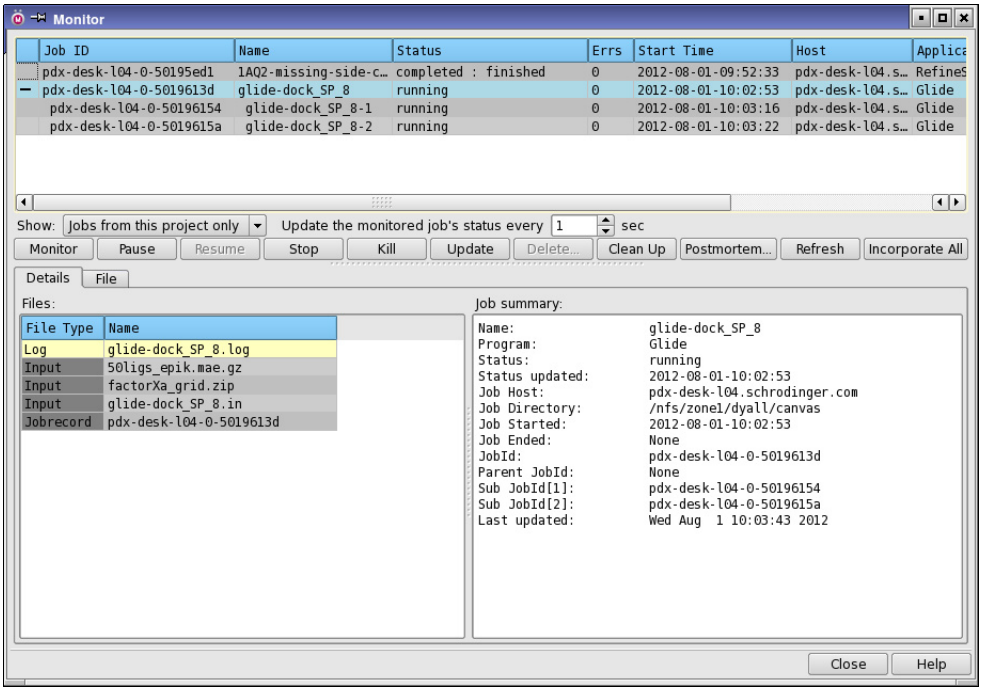

Figure 3.1. The Monitor panel, showing the Details tab.

at the end of the file. If the display is not at the end of the file when text is added, the display remains at the same location in the file, and you must scroll down to the end of the file to see the new text.

3.3 Managing Jobs From the Command Line

The job control utility allows you to perform a number of job control tasks from the command line. The syntax for job control utility commands is:

```
$SCHRODINGER/jobcontrol action [options] query
```

where *action* is the command for the action you want to perform, *options* qualifies the scope of the command, and *query* defines the scope of the action performed by the command. Valid commands are listed in Table 3.5, and valid options are listed in Table 3.6.

Table 3.5. Actions for the jobcontrol command.

Action	Description
`-abort`	Emergency stop, abandon output files and the job record.
`-cancel`	Cancel a job that has been launched, but not started.
`-delete [-force]`	Remove a completed job from the database. `-force` removes any job from the database, whether completed or not, and should only be used when other methods fail.
`-dump`	Show the complete job record.
`-g[ui]`	Start the Job Control web interface in a browser.
`-h[elp]`	Describes the job control commands.
`-int` *seconds*	Interval for checking job status (default: 5 seconds).
`-kill`	Terminate the job immediately.
`-killnooutput`	Terminate the job immediately and discard the output.
`-list`	List the JobId, job name and status. Include subjobs if `-children` is used.
`-loglevel` *level*	Set the `jmonitor` log level. Allowed values: `none`, `log`, `debug`.
`-monitor` *n*	Ask for monitoring files to be sent every *n* seconds. A value of 0 turns off monitoring.
`-pause`	Suspend the job temporarily.
`-ping`	Verify that a running job responds to job control messages.
`-recover`	Try to retrieve results from a job that did not finish successfully.

Table 3.5. Actions for the jobcontrol command. (Continued)

Action	Description
`-refresh`	Update the status of the job.
`-resume`	Continue running a paused job.
`-show`	Show more details of the job status, including the command used to run the job.
`-stop`	Ask the job to stop itself as soon as possible (Jaguar, MacroModel only)
`-update`	Ask for an update of the job results. (MacroModel only)
`-v[ersion]`	Display program version and exit.
`-view [-l` *log*`]`	View the job log files in real time.
`-wait`	Wait for the job to finish before returning to the command prompt.

Table 3.6. Options for the jobcontrol command

Option	Description
`-c[hildren]`	Include the subjobs of the selected jobs. For the `-list` action, list all subjobs individually.
`-noc[hildren]`	Don't include the children of the selected jobs. For the `-list` action, don't include subjob summaries.
`-m[issing]`	Include jobs with missing job records, for which only auxiliary files are present.
`-d[ebug]`	Show debug information when performing this action, even if running in non-debug mode.
`-n[odebug]`	Do not show debug information, even if running in debug mode.
`-f[orce]`	Used with `-delete` to force removal of job record.
`-l[og]` *file*	Used with `-view`, specifies which log files to view

The optional *query* consists of one or more JobIds, job names, status codes or queries, or the keywords `all`, `active`, or `last`*N*. The default *query* is all active jobs (jobs that are not finished), and is equivalent to using the keyword `active`. The keyword `all` means all jobs in your job database. The keyword `last`*N* means the *N* most recently submitted jobs; `last` is equivalent to `last1`. The JobId is a unique identifier consisting of the name of the submission host, a sequence number, and a hexadecimal timestamp, e.g., `mirabelle-0-a1b2c3d4`. The job record fields and their meanings are listed in Table 3.1 on page 24.

The following examples illustrate different values of *query*.

- To list all active jobs, showing their JobIds, job names, current status and the machine on which each is running, enter:

```
$SCHRODINGER/jobcontrol -list
```

- To list all the jobs in your job database that finished successfully, enter:

```
$SCHRODINGER/jobcontrol -list finished
```

- To list just the job whose JobId is `serv01-0-a1b2c3d4`, enter:

```
$SCHRODINGER/jobcontrol -list serv01-0-a1b2c3d4
```

- To list all jobs in your database with the job name `myjob`, enter:

```
$SCHRODINGER/jobcontrol -list myjob
```

- To list all jobs in your database, enter:

```
$SCHRODINGER/jobcontrol -list all
```

- To list the 5 jobs most recently submitted, enter:

```
$SCHRODINGER/jobcontrol -list last5
```

You can also use the wildcard character '?' to match a single unspecified character, or '*' to match zero or more unspecified characters. If you use either of these characters, you must protect them to ensure that they are interpreted by the `jobcontrol` script and not the UNIX shell. For example, you could enter either of the following commands to list all jobs whose job names start with `docklig`

```
$SCHRODINGER/jobcontrol -list docklig\*
$SCHRODINGER/jobcontrol -list 'docklig*'
```

3.3.1 General Job Control Queries

More general queries than those given above are also possible. Formally, a query consists of one or more *conditions*, optionally separated by the Boolean operators AND, OR or NOT. If the operators are omitted, OR is assumed. A condition takes the form <field><op><value>, where <field> is one of the field names in the job record and <op> is one of the following:

=	equals
!=	is not equal to
=~	matches
!~	does not match

"Equals" means an exact match; "matches" means that <field> matches <value>, treated as a regular expression.

The field names are listed in Table 3.1 on page 24. The fields most likely to be useful for queries are Name, Program, Host, Dir, JobHost, JobDir, Project, Status, and ExitStatus. The case of the field names is ignored, but the case of the <value> is significant. So, `Program=Jaguar` is the same as `program=Jaguar`, but `program=jaguar` would fail. Parentheses can be used to group conditions, but these must also be protected if used on the Unix command line.

For example, to list all QSite jobs in your database that either died or were killed, enter the command:

```
$SCHRODINGER/jobcontrol -list program=QSite AND
    \( died OR killed \)
```

3.3.2 Recovering Stranded Jobs

Job Control periodically tries to re-establish contact with stranded jobs. It is also possible to re-establish contact with the job manually, with the command:

```
$SCHRODINGER/jobcontrol -recover jobid
```

If this command fails, however, the job is cleaned up and the results are no longer available.

If you cannot recover a job you can remove the job record with the command:

```
$SCHRODINGER/jobcontrol -delete -force jobid
```

This command should only be used as a last resort, because it removes all job control information for the given job. However, if the job is actually alive, its job record may eventually reappear after using this command.

3.3.3 Using jserver

The `jserver` process on the submission host sets up and manages the connection to remote hosts. Normally it is started automatically. If it has stopped, you can restart it on a UNIX system with the following command:

```
$SCHRODINGER/utilities/jserver [-proxy] [action] [options]
```

The actions for this command are given in Table 3.7, and the options are given in Table 3.8. The options only apply to newly started `jserver` processes.

In addition to starting a new `jserver` process, the `jserver` command can be used to interact with a currently running `jserver` process. If a new `jserver` process is started without the

-force action, it checks whether another jserver process is already running. If so, the new process contacts the running process to determine which process has the latest version. If the new process has a later version, it kills the old jserver process and replaces it; otherwise, the new process exits.

A jserver process started with the -proxy option is referred to as a jproxy process. These processes are used for batch jobs to mediate communications between the job and the jserver process running on the launch host. A user can have at most one jserver process and one jproxy process running on a given host.

If jserver stops running, the jobs it manages are not incorporated until jserver restarts. Then the remote job can copy files back to the submission host. To check if jserver is running, use the -info option. You can also kill jserver with the -kill option, and you can kill jserver and start a new jserver with the -force option.

On Windows, jserver is automatically restarted when you start your computer.

Table 3.7. Actions for the jserver command

Action	Description
-clean	Shut down jserver or jproxy and delete all related files (log, state, PID, plus jnanny files for jserver but not jproxy).
-force	Start a new jserver process after killing any active jserver process.
-help	Display the usage message.
-info	Report whether a jserver process is already running
-kill	Kill the active jserver process and exit.
-loglevel *level*	Change the verbosity level for the jserver log file. Allowed values: quiet, log, debug, or trace.
-se[nd] *message*	Send the specified message to jserver and show the response.
-show *object*	Report on various objects: jobs—show which jobs jserver is managing. jproxys—show the jproxy processes jserver is communicating with. jservers—show the jserver processes jproxy is communicating with. sockets—show information on the sockets used by jproxy.
-shutdown	Shut down the entire jserver/jproxy network.
-version	Report the version of the jserver process that is running.

Table 3.8. Options for the jserver command.

Option	Description
`-port` *list*	Specify the list of ports to use for the server. The list is comma-separated, and can include ranges given either by *m*`:`*n* or *m*`-`*n*. The list must not contain spaces.
`-output`	Run jserver in the foreground and write log messages to `stdout`.
`-removepid`	Remove pidfile upon exit.
`-quiet`	Run in quiet mode (no output except error messages).
`-verbose`	Run in verbose mode (default).
`-debug`	Run in debug mode (extra output, useful for debugging).

The `jserver` process writes to a log file named `jserver.`*hostname*`.log` in the job database directory. You can limit the size of this file by setting the environment variable `SCHRODINGER_JSERVER_LOGSIZE`—see Appendix B for details.

3.3.4 Purging the Job Database

Normally, when running jobs from Maestro's project facility, the database record for a job is cleared once the job is incorporated into a Maestro project. Not all jobs can be incorporated, however; for this and other reasons job records might not get deleted automatically and the job database directory can become quite large over time. The `jobcontrol` utility can be used to purge the job database of records for completed jobs. For example, the following command purges the entire database:

```
$SCHRODINGER/jobcontrol -delete all
```

This command only affects completed jobs: running jobs cannot be deleted unless `-force` is supplied after `-delete`. However, you should be careful not to delete completed jobs whose output you still intend to incorporate into a Maestro project.

The job database is periodically checked for jobs that have finished by a process called `jnanny` (see Section 3.3.5 on page 36). If the job is older than a threshold time, it is deleted from the database. You can set this threshold time in the environment variable `SCHRODINGER_JOBDB_CLEANUP`. The default time is 1 week; the minimum allowed time is 1 second, but a realistic minimum is 10 minutes. The default unit is seconds, but you can specify a time in minutes, hours, or days by appending m, h, or d to the value, for example, 7d, 168h, or 5m.

3.3.5 Performing Actions on Jobs with jnanny

The `jnanny` script scans the job database and performs tasks defined by plugin modules. These tasks include checking for jobs that are stranded or appear to be stuck in a particular state, and then initiating a recovery process and updating the job database. This script is run periodically by `jserver`. The interval at which an action is performed can be defined by the environment variables listed in Table 3.9. The default unit is seconds, but you can specify a time in minutes, hours, or days by appending m, h, or d to the value.

Since `jnanny` is run periodically, it should not in general be necessary to run it manually. However, you can do so if the need arises, with the following command:

```
$SCHRODINGER/utilities/jnanny [options] [plugins]
```

The options are described in Table 3.10, and the available plugins are described in Table 3.11. If no plugins are specified, all plugins are called. You can abbreviate a plugin specification to the shortest unique string.

Table 3.9. Environment variables that define interval of jnanny actions.

Environment variable	Description
`SCHRODINGER_STRANDED_THRESHOLD`	Time interval for checking for stranded jobs. Default: 20 minutes.
`SCHRODINGER_JOBDB_CLEANUP`	Time interval for cleaning up the job database. Default: 1 week (see Section 3.3.4 on page 35).
`SCHRODINGER_SUBMITTED_THRESHOLD`	Time interval for checking on submitted and launched jobs to ensure that they are progressing. Default: 10 minutes.

Table 3.10. Options for the jnanny command

Option	Description
`-d[ebug]`	Run in debug mode.
`-q[uiet]`	Run quietly (do not generate any output).
`-a[ll]`	Run all plugins.
`-l[ist]`	List the available plugins.
`-norun`	Show plugins to run but do not actually run them.
`-s[tatus]`	Report on the status of each plugin.

Table 3.11. Available plugins for jnanny.

Plugin	Description
`[-]CheckStranded`	Checks for stranded jobs and initiates recovery with the command `jobcontrol -recover`. Run automatically every 20 minutes while there are active jobs.
`[-]CheckSubmitted`	Checks launched and submitted jobs to ensure that they respond to `jobcontrol` commands, using `jobcontrol -ping`. Run automatically every 10 minutes while there are launched and submitted jobs.
`[-]CleanupDatabase`	Cleans up the job database, by deleting records for incorporated jobs and finished, unincorporatable jobs that are older than the cleanup time. Run automatically every 10 minutes, or if needed.
`[-]MonitorJobs`	Communicates to jserver which jobs it is responsible for. Run automatically when jserver first starts up; rerunning it manually has no effect.
`[-]UpdateDbIndex`	Compact the job index and ensures that it is compatible with the database. Most ordinary operations only append to the index without removing invalidated records. Run automatically every 10 minutes.

The postmortem Utility

The `postmortem` utility archives information that is useful for understanding why jobs do not run as expected. It creates a tar archive containing the current environment, file system information, a list of installed Schrodinger software packages, the `schrodinger.hosts` file, the queue support scripts, the license file, and information such as input and output for specified jobs. When contacting technical support regarding a job, it is highly recommended to run this utility, and send the archive with your email message. It is also highly recommended that you generate this archive using the Diagnostics panel, which runs the `postmortem` utility and creates the archive in a form that is accepted by mail programs.

Note: You should ensure that no sensitive information is added to the archive, by using the options described below, and by inspecting the contents of the archive.

The syntax of the `postmortem` utility is as follows:

`$SCHRODINGER/utilities/postmortem` [*options*] [*jobids*]

The options are described in Table A.1. Options may be truncated to any unique abbreviation.

Table A.1. Options for the postmortem command.

Option	Description
`-autoreplace`	Automatically replace strings that may contain sensitive information.
`-exclude` *suff*	Do not include files with the given suffix in the archive.
`-help`	Show the usage message.
`-nojobfiles`	Do not include any input, output or log files in the archive.
`-noparents`	Do not include parent jobs of the specified jobs in the archive.
`-nosubjobs`	Do not include subjobs of the specified jobs in the archive.
`-output` *name*	The base name of the archive: the archive is named *name*`.tar.gz`. The default is *username*`-`*host*`-schrodinger.tar.gz` if no job IDs are given, and *jobid1*`-archive.tar.gz` if job IDs are given, with *jobid1* the first job ID.
`-quiet`	Do not print anything on standard output.
`-replace` *string*	Replace the given string in the archived files.

Table A.1. Options for the postmortem command.

Option	Description
`-size`	Calculate the total size of the job files and exit; do not create an archive.
`-struct`	Include structure files in the archive. These files are excluded by default.
`-verbose`	List the names of the archived files on standard output.

If job IDs are specified, the job records for those jobs are archived, along with any related files (batch scripts, qlog files, and so on) from the job database. Parent and subjobs of the specified jobs are included as well, unless the option `-noparents` or `-nosubjobs` is given. Finally, unless `-nojobfiles` is given, any log files and non-structural input or output files listed in the job record are saved as well. If you want to include structure files in the output, use the `-struct` option.

The archive can be quite large if all job files for a large job are included. You might want to use the `-size` option to check the total size of the files to be archived, first.

To find the job IDs for your jobs, use the `jobcontrol` command, or look in the Monitor panel.

For example the following command lists all of your completed jobs:

```
$SCHRODINGER/jobcontrol -list completed
```

If file and directory names contain sensitive information that you don't want to reveal, you can use the `-autoreplace` option to have the program replace them in the archived files. You can specify particular string replacements using the `-replace` option, as well. A list of all string replacements that were done is written to a file called *archive*`.names`.

Environment Variables

There are a number of environment variables that can be used to specify the location of resources, manage the job database, and control various aspects of job execution. These environment variables are given in Table B.1. Some of these environment variables provide alternate means of specifying the resource; others are the sole means of specifying a resource or of overriding a default value. For information on setting environment variables, see Appendix A of the *Installation Guide*.

Table B.1. Environment variables for resource specification and job control.

Variable	Purpose
SCHROD_LICENSE_FILE LM_LICENSE_FILE	Specify the license file or a list of license files to use if the license file is not installed in $SCHRODINGER. The list must be separated by colons on Unix and semicolons on Windows. The SCHROD_LICENSE_FILE definition supersedes any other definition. Both of these environment variables are FlexLM environment variables. Use SCHROD_LICENSE_FILE when LM_LICENSE_FILE does not point to a license file containing Schrödinger licenses. See Section 6.5 of the *Installation Guide*.
SCHRODINGER	Specify the installation directory.
SCHRODINGER_AUTO_SAVE	If set to 1, force the contents of the job directory to be archived and copied back to the submission host only for those jobs that have died (whose exit status is "died"). It is also possible to force such copying for all jobs (see SCHRODINGER_SAVE_JOBDIR).
SCHRODINGER_HOSTS	Specify the hosts file. See Section 2.3.5 on page 15.
SCHRODINGER_HUNT_CACHE_EXPIRY	Time interval for the caching of installation data collected by hunt to expire. Can be set to a time with the suffix d, h, m, or s. Default: 24h.
SCHRODINGER_JOB_DEBUG	Control debugging output. Allowed values are: 0 No debugging output 1 Standard debugging output (same as entering -DEBUG from the command line) 2 Detailed debugging output (same as entering -DDEBUG from the command line)

Table B.1. Environment variables for resource specification and job control. (Continued)

Variable	Purpose
SCHRODINGER_JOB_DISPOSITION	Specify how to incorporate job results into the project (same as entering -DISP from the command line) Available values are: append—add entries to project in an entry group appendungrouped—add entries to project singly replace—replace project entries with new entries ignore—do not incorporate entries Default: ignore See Section 2.9 on page 19.
SCHRODINGER_JOBDB2	Specify the full path name for the job database. Must not be set to a location that contains a pre-2011 job database. Note that SCHRODINGER_JOBDB is no longer supported, and will be ignored. Default: ~/.schrodinger/.jobdb2 See Section 3.1 on page 23.
SCHRODINGER_JOBDB_CLEANUP	Specify how long the job record of a completed job is kept in the job database before being automatically deleted. For example, one day can be expressed as 86400, 1440m, 24h, or 1d. Default: 7 days. See Section 3.3.4 on page 35.
SCHRODINGER_JOBDB_MAXSIZE	Specify the maximum number of job records to be kept in the job database. If this number is exceeded, job records of completed jobs are purged, starting with the oldest, at the next cleanup time. Default: 2000 records.
SCHRODINGER_JMONITOR_PORT	Specify the port or range of ports that jmonitor may use. Ports can be specified as comma or colon-separated lists without spaces. Ranges can specified with a dash, for example, 5987:5989-5992:5994. Set this variable if there is a firewall that allows connections to ports within a given range only. The number of ports should be twice the number of jmonitor processes (jobs).
SCHRODINGER_JPROXY	Regulate the use of jproxy for the job. If set to 1, jproxy is used; if set to 0, it is not. By default (if not set), jproxy is used for batch jobs.

Table B.1. Environment variables for resource specification and job control. (Continued)

Variable	Purpose
SCHRODINGER_JPROXY_PORT	Specify the port or range of ports that `jproxy` may use. Overrides any setting of `proxyport` in the hosts file. The syntax is the same as for `SCHRODINGER_JSERVER_PORT`. If this environment variable is not set, the values set in `SCHRODINGER_JSERVER_PORT` apply to `jproxy` as well. Therefore, it is generally not advisable to specify only one port in `SCHRODINGER_JSERVER_PORT`, as this will make it impossible for `jserver` and `jproxy` to run on the same machine, which may be necessary.
SCHRODINGER_JSERVER	Regulate the use of `jserver` for the job. If set to 1, `jserver` is used; if set to 0, it is not. By default (if not set), `jserver` is used for remote and batch jobs.
SCHRODINGER_JSERVER_LOGSIZE	Set the size threshold for the `jserver` log file in bytes. Units of kilobytes and megabytes can be used by appending `K` or `M` to the value. The five most recent log files are kept, with extensions `.1` through `.5`. When the file size exceeds this threshold, the files are rotated, the `.log.5` file is discarded if it exists, and a new log file is started.
SCHRODINGER_JSERVER_PORT	Specify the port or range of ports that `jserver` may use. Ports can be specified as comma or colon-separated lists without spaces. Ranges can specified with a dash, for example, `5987:5989-5992:5994`. Setting this variable is useful if there is a firewall that allows connections to ports within a given range only.
SCHRODINGER_LICENSE_DEBUG	Turns on debugging related to license usage. Any non-zero integer value turns on debugging.
SCHRODINGER_LICENSE_RETRY	Specify the maximum time to keep trying to obtain a license. The value can be an integer value in seconds, or an integer value with a time unit, such as 7200s, 120m, 2h. Default: 10 minutes.
SCHRODINGER_LOGFILE_CLEANUP	Clean up (remove) `jserver` and `jproxy` log files that are older than the specified age. The value can be an integer value in seconds, or an integer value with a time unit, such as 7200s, 120m, 2h. Default: 1 month.

Table B.1. Environment variables for resource specification and job control. (Continued)

Variable	Purpose
SCHRODINGER_MAX_RETRIES	Specify the number of times a failed subjob from a distributed job is rerun before exiting with a failure. Only supported for distributed Glide and workflow jobs. Default: 3.
SCHRODINGER_MONITOR_INTERVAL	Set the interval at which monitored files are copied back to the launch directory.
SCHRODINGER_NICE	Lower the priority of Schrödinger jobs. If it is unset or set to a null string, the priority is not lowered. The value `local` lowers the priority on jobs and subjobs submitted to the local host, but does not lower the priority of jobs or subjobs submitted to a remote host. Any other value lowers the priority of all jobs. By default, most jobs run on the local host from Maestro are run with reduced priority. To change this, set this variable to a null string. Equivalent to using the `-NICE` command-line option.
SCHRODINGER_NO_HUNT_CACHE	Turn off the caching of installation data by `hunt`, by setting this variable to any non-empty value except 0. If turned off, `hunt` contacts the remote host when a job is launched to find compatible software; if caching is on, it checks a locally cached list of compatible software.
SCHRODINGER_NO_NFS	If set, avoid using NFS mounts for file transfers between the job directory and the submission (output) directory, if possible.
SCHRODINGER_NODEFILE	Specify the path to a file containing a list of hosts and the number of CPUs to use on each host. Each line in the file must have the format: *host*:*numberofCPUs* You can list a host multiple times rather than provide the number of CPUs; this may affect the execution of parallel jobs. This environment variable is used to pass an explicit list of hosts to the driver for a parallel or distributed job. It is not used by Job Control directly; as such, the meaning of the file depends on the product that uses it. For a parallel job, such as parallel Jaguar or Desmond, the names in the file must be actual host names, not entries in the hosts file. For distributed jobs, like Glide docking, the names must be entries in the hosts file, since they are used for launching subjobs.

Table B.1. Environment variables for resource specification and job control. (Continued)

Variable	Purpose
	Ordinarily, the hosts to use for a distributed or parallel job would be supplied using command line arguments, as explained elsewhere in this document. This variable is provided to allow the usual mechanism to be overridden when necessary.
SCHRODINGER_PROJECT	Specify the path name of the Maestro project into which the results of the job should be incorporated (same as entering -PROJ from the command line) See Section 2.9 on page 19.
SCHRODINGER_QLOGDIR	Specify the path to the directory where queue log files (.qlog) for batch jobs are to be written. The default is the job database directory; however, if that directory is not mounted on the queue host, an alternative path needs to be specified, or the queueing system will not be able to write those log files.
SCHRODINGER_QUEUE_ARGS	Specify queue arguments for job submission to a queueing system. These arguments are also used when subjobs are submitted by a master job. Queue arguments specified with this environment variable are appended to those read from schrodinger.hosts.
SCHRODINGER_RETAIN_JOBDIR	If set to 1, when the job finishes and the output files are copied back, the job directory is not removed. By default, it is removed if it did not exist already.
SCHRODINGER_RSH	Specify the command to use for remote connection (see Section 7.2 of the *Installation Guide*). You can use rsh or ssh, if they are specified in your path, otherwise you must use the full path name to the command. The default, if SCHRODINGER_RSH is not specified, is ssh on UNIX and plink.exe on Windows.
SCHRODINGER_SAVE_JOBDIR	If set to 1, archive the contents of the job directory and copy it back to the submission host when the job finishes. The archive appears in the output directory as *jobid*-*jobdir*.zip. It is also possible to force such copying only for jobs that have died (see SCHRODINGER_AUTO_SAVE). Equivalent to using the -SAVE command-line option.

Table B.1. Environment variables for resource specification and job control. (Continued)

Variable	Purpose
SCHRODINGER_SECURE_TRANSFERS	Enforce or cancel data transfer via ssh tunnels. If set to 1, secure transfers are used regardless of any secure zone settings in the hosts file. If set to 0, secure transfers are not used at all. If unset, secure transfers are used for transfers outside the secure zones defined in the hosts file.
SCHRODINGER_STRANDED_THRESHOLD	Time interval for checking for stranded jobs by jnanny. Default: 20 minutes.
SCHRODINGER_SUBMITTED_THRESHOLD	Time interval for checking on submitted and launched jobs by jnanny to ensure that they are progressing. Default: 5 minutes.
SCHRODINGER_TMPDIR	Specify the full path name for the directory in which run time temporary job directories are created. Overrides the tmpdir setting specified in the schrodinger.hosts file (Section 7.1.3 of the *Installation Guide*).
SCHRODINGER_VER_ARGS	Specify version options to be used when submitting a job. The value must be a list of options as they would appear on the command line, taken from the options listed in Table 2.4 on page 11.

Secure File Transfer to Queue Hosts

All file transfers made to and from queue hosts for jobs submitted to a batch queue are secured by default, so that sensitive information can be transmitted across insecure networks. The core of the secure transfer mechanism is an SSH tunnel ("secure channel") established between `jserver` on the launch host and `jproxy` on the queue manager host. The SSH tunnel is established in the background, automatically, in a transparent manner, and is removed when it is no longer needed. This ensures that all data transfers between the launch host and the computing cluster are encrypted. Communication between the queue manager host and compute nodes is not secured, as the main goal of the secure file transfer mechanism is to protect information as it enters and leaves the queue host.

Simple remote jobs from one host to another, which do not involve a batch queue, are never secured by the secure transfer mechanism.

The most common use case for a secure transfer channel involves a workstation that communicates with the head node of a cluster or the Cloud via the internet. In this case, it is not desirable to transfer data in plain text, since it would be exposed over the internet. It is common for head nodes in such situations to employ firewalls that block all incoming connections except for SSH. The secure transfer mechanism is well suited for both of these concerns.

If the connection between the job submission host and the queue host is already secure (for example, part of a secure network), there is no need for the secure file transfer mechanism. In this situation secure transfers can be disabled altogether or within specified *secure zones*.

C.1 Controlling Secure File Transfer

Secure file transfers can be disabled between a specific set of hosts by defining secure zones in the hosts file. See Section 7.1.9 of the *Installation Guide* for information on defining these zones.

While it is possible for a host to be in more than one secure zone, hosts never act as bridges between different zones. For example, if one zone includes host1 and host2 and another one includes host2 and host3, then connections between host2 and host1 or host3 are not secured, but connections between host1 and host3 are secured because they do not belong to the same secure zone. The file transfers do not take place via host2, so it does not act as a bridge between the two secure zones.

Secure file transfers can be turned on and off globally by setting the environment variable SCHRODINGER_SECURE_TRANSFERS. A value of 0 turns off secure file transfers, and a value of 1 turns on secure file transfers to all queue hosts. This setting takes precedence over all other settings, including secure zones. Secure zones are used only if this variable is undefined.

The active secure zone information is stored in the job database directory, in the same format as in the hosts file:

~/.schrodinger/.jobdb2/securezones.*host*

C.2 Altering Secure Zone Settings

Since secure zones are interpreted and used by jserver, which is shared between all of the jobs that you launched from a given host, you cannot change the secure zone configuration while jserver is active. If you do, jobs launched after such a change fail with an error message: "Conflicting secure zones information". If you do need to change the secure zone configuration, you must kill your current jserver, with the command

```
$SCHRODINGER/utilities/jserver -clean
```

and then make changes to the securezone entries of your hosts file before launching any subsequent jobs. If the hosts file is on a network file system, all jserver processes that have access to that file system must be killed.

The same applies to the SCHRODINGER_SECURE_TRANSFERS environment variable. If you need to change its value (including undefining it if it was defined previously), you must kill jserver before launching any further jobs.

C.3 Verifying Secure Transfer

To verify whether connections between a jserver process and one or more of jproxy processes are secure, use the command

```
$SCHRODINGER/utilities/jserver -show jproxys
```

The output should be something like the following:

```
Found jserver (PID 1645) listening on port 47129
monitoring 2 jproxy processes
untrustedhost [JPROXY_OK (secure)]
trustedhost [JPROXY_OK]
```

In this example, the connection between your host and untrustedhost is secured, while that between your host and trustedhost is not.

You can also use the Diagnostics GUI to determine whether or not jobs sent to a particular entry in the hosts file is secured. In the GUI, select the desired host from the Hosts table, and use Selected Host to run a test job. If the `testapp` result is `OK (SECURE)`, then jobs sent to that cluster entry will use the secure transfer mechanism.

Getting Help

Schrödinger software is distributed with documentation in PDF format. If the documentation is not installed in `$SCHRODINGER/docs` on a computer that you have access to, you should install it or ask your system administrator to install it.

For help installing and setting up licenses for Schrödinger software and installing documentation, see the *Installation Guide*. For information on a particular product, see the documentation for that product. For information on using Maestro and its panels, see the Maestro online help or the *Maestro User Manual*. Other information is available on the Schrödinger Support Center and the knowledge base, http://www.schrodinger.com/kb.

Contacting Technical Support

If you have questions that are not answered from any of the above sources, contact Schrödinger using the information below.

E-mail:	help@schrodinger.com
USPS:	Schrödinger, 101 SW Main Street, Suite 1300, Portland, OR 97204
Phone:	(503) 299-1150
Fax:	(503) 299-4532
WWW:	http://www.schrodinger.com
FTP:	ftp://ftp.schrodinger.com

Generally, e-mail correspondence is best because you can send machine output, if necessary. When sending e-mail messages, please include the following information:

- All relevant user input and machine output
- Schrödinger Suite purchaser (company, research institution, or individual)
- Primary Schrödinger Suite user
- Installation, licensing, and machine information as described below.

Gathering Information for Technical Support

This section describes how to gather the required machine, licensing, and installation information, and any other job-related or failure-related information, to send to technical support.

For general enquiries or problems:

1. Open the Diagnostics panel.
   - **Maestro:** Help → Diagnostics
   - **Windows:** Start → All Programs → Schrodinger-2012 → Diagnostics
   - **Mac:** Applications → Schrodinger2012 → Diagnostics
   - **Command line:** `$SCHRODINGER/diagnostics`
2. When the diagnostics have run, click Technical Support.

   A dialog box opens, with instructions. You can highlight and copy the name of the file.
3. Attach the file specified in the dialog box to your e-mail message.

If your job failed:

1. Open the Monitor panel in Maestro.

   Use Applications → Monitor Jobs or Tasks → Monitor Jobs.
2. Select the failed job in the table, and click Postmortem.

   The Postmortem panel opens.
3. If your data is not sensitive and you can send it, select Include structures and deselect Automatically obfuscate path names.
4. Click Create.

   An archive file is created in your working directory, and an information dialog box with the name of the file opens. You can highlight and copy the name of the file.
5. Attach the file specified in the dialog box to your e-mail message.
6. Copy and paste any log messages from the window used to start Maestro (or the job) into the email message,or attach them as a file.
   - **Windows:** Right-click in the window and choose Select All, then press ENTER to copy the text.
   - **Mac:** Start the Console application (Applications → Utilities), filter on the application that you used to start the job (Maestro, BioLuminate, Elements), copy the text.

If Maestro failed:

1. Open the Diagnostics panel.
   - **Windows:** Start → All Programs → Schrodinger-2012 → Diagnostics
   - **Mac:** Applications → Schrodinger2012 → Diagnostics
   - **Linux/command line:** `$SCHRODINGER/diagnostics`

2. When the diagnostics have run, click Technical Support.

   A dialog box opens, with instructions. You can highlight and copy the name of the file.

3. Attach the file specified in the dialog box to your e-mail message.

4. Attach the file `maestro_error.txt` to your e-mail message.

   This file should be in the following location:

   - **Windows:** `%LOCALAPPDATA%\Schrodinger\appcrash`
     (Choose Start → Run and paste this location into the Open text box.)
   - **Mac:** `Documents/Schrodinger`
   - **Linux:** Maestro's working directory specified in the dialog box (the location is given in the terminal window).

5. On Windows, also attach the file `maestro.EXE.dmp`, which is in the same location as `maestro_error.txt`.

Glossary

entry—(1) A collection of settings in the hosts file that defines a configuration for running jobs on a given host. There can be more than one entry for a given host, with different settings. (2) A structure and its properties in a Maestro project.

execution host —The computer that a job runs on.

hosts file—The file that contains information on available hosts that is used by Job Control. This file is usually named `schrodinger.hosts`.

job directory—The directory on the execution host to which the input files are copied from the submission host and from which the output files back to the submission host when the job is finished. This is usually a subdirectory of the scratch directory that is created by Job Control.

launch directory—The directory from which the job is started. Same as *submission directory.*

launch host—The computer that you submit a job from. Same as *submission host*.

local host—The computer that you are logged on to. Usually this is also the computer you are using for job submission.

output directory—The directory to which output files are copied at the end of the job. This is usually the same as the submission directory.

queue host—A computer that can run the queueing software. This is not necessarily the queue manager: for example, if the queuing software is installed on an NFS-mounted file system, it could be any host that has that file system.

remote host—A computer that is available to you over a network.

remote job submission—Submitting a job to a computer other than the one you are logged on to or that you are running Maestro on.

Schrödinger user resource area—location for storage of resources (files, scripts, data) used by Schrödinger software for an individual user. This location is `$HOME/.schrodinger` on Linux and `%APPDATA%\Schrodinger` on Windows.

scratch directory—The directory that is used for temporary files. The files are usually created in a subdirectory of this directory that is unique to the user and the job.

submission directory—The directory from which the job is started. This is the directory from which input files (specified without a path) are copied to the job directory, and to which output files are copied at the end of the job.

submission host—The computer that you submit a job from.

Index

L

M

O

P

Q

R

S

T

U

W

120 West 45th Street
17th Floor
New York, NY 10036

101 SW Main Street
Suite 1300
Portland, OR 97204

245 First Street
Riverview II, 18th Floor
Cambridge, MA 02142

8910 University Center Lane
Suite 270
San Diego, CA 92122

155 Gibbs St
Suite 430
Rockville, MD 20850-0353

Dynamostraße 13
D-68165 Mannheim
Germany

Zeppelinstraße 73
D-81669 München
Germany

Potsdamer Platz 11
D-10785 Berlin
Germany

Quatro House
Frimley Road
Camberley GU16 7ER
United Kingdom

8F Pacific Century Place
1-11-1 Marunouchi
Chiyoda-ku, Tokyo 100-6208
Japan

No. 102, 4th Block
3rd Main Road, 3rd Stage
Sharada Colony
Basaveshwaranagar
Bangalore 560079, India

SCHRÖDINGER®