Daylight Chemical Information Systems, Inc.
120 Vantis - Suite 550 - Aliso Viejo, CA 92656
tel +1 949-831-9990 - fax +1 949-831-9902 - www.daylight.com

Daylight THOR-Merlin Manual

Table of Contents

Table of Contents

Daylight THOR-Merlin Manual

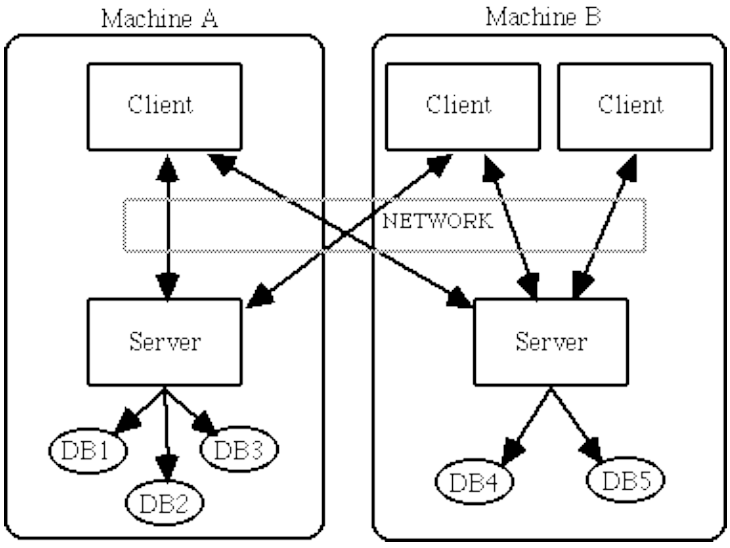
Daylight Version 4.9
Release Date 08/01/11

1. Networking

THOR and Merlin are powerful tools for maintaining and exploring chemical databases. In this chapter, we explore the client/server network architecture that THOR and Merlin use; the chapters that follow describe the chemical-information aspects of THOR and Merlin. Both THOR and Merlin are client/server systems and the following will be applicable to both; where there are differences it will be noted. In general, a typical Daylight database system will involve the two servers, thorserver and merlinserver, and one or more client program which connects to one or both types of servers (e.g., XVMerlin is both a thorserver and merlinserver client).

As the name implies, a client/server system actually runs as two programs: a client (an application program), and a server (a program supplied by Daylight). The words "client" and "server" give you some idea about the relationship between the two: The server provides a service (database access), and the client uses that service.

The server's role is relatively straightforward: It has exclusive access to all databases that it opens, and performs all database transactions. Clients never access a database directly, but rather access all data via a server. The server can serve many clients at once, so normally only one server runs at a time on each computer, no matter how many clients are running.



The client's job is somewhat more complex. Clients have to deal with users (who are notoriously unpredictable), communicate with the server, generate unique SMILES for use with the server, and other such tasks. Each client can connect to many servers, and each client can open many databases on each of the

servers.

Normally the client/server design is nearly transparent to the application. Given a server's "name and address" (using standard network protocol), the Daylight system takes care of connecting to the server via an Interprocess Communication (IPC) mechanism, and handles all communication with the server. When a client requests a database transaction, the Daylight system translates it into a request to the server; the message is sent via IPC, the server responds via IPC, and the Daylight system translates the response back into the answer to the client's request.

Because clients and servers communicate via network protocol, the client and server can reside on different machines or on the same machine. The IPC mechanism is designed to work equally well either way; once a connection to a server is established (a procedure that is the same no matter where the server resides), the client doesn't have to be concerned about the server's actual location.

The client/server mechanism uses a standard networking mechanism to communicate, one that is independent of the operating system of the machines on which the client and server are located. This means that heterogeneous mixtures of machines can be used to run a Daylight database system. For example, a large Sun system might run a server program; this server could serve a number of SGI workstations situated on a local-area network (LAN).

Some other advantages of the client/server design are:

- **Guaranteed uninterruptible operations** The server completes each transaction before it responds to a request from another client. There is no need for file locking, record locking, or other exclusive-access mechanisms; only one program ever accesses a database.
- **Networkable** The server can be placed on a big, fast machine with googlebytes of disk and memory, serving smaller workstations around a laboratory (or at remote sites if a wide-area network is available). The cost of expensive equipment (the disk, memory, and high-speed computer), as well as valuable data, can be shared among many clients.
- **Caching** With a single server, it is possible to cache (store in memory) parts that are accessed frequently. Caching is nearly impossible in "traditional" systems where multiple programs have to simultaneously access a database, due to synchronization problems.
- **Access control** The server can, at any time, list the users of a particular database. It can also lock a database for exclusive access even when other users already have it open, and other such functions that require complete control over who is using a database.
- **Database integrity** The server is a relatively simple, robust program. When a client program has a problem, such as crashing due to a programming errors during debugging, the server detects the problem and simply closes the database. Similarly, because all requests go through the server, which verifies that each transaction is legitimate before executing it, it is virtually impossible for a client to corrupt a database.
- **Security** Only the server process has direct access to databases. Security is greatly simplified and improved since all client programs must pass the server's scrutiny.

Of course, the client/server mechanisms do carry a modest price:

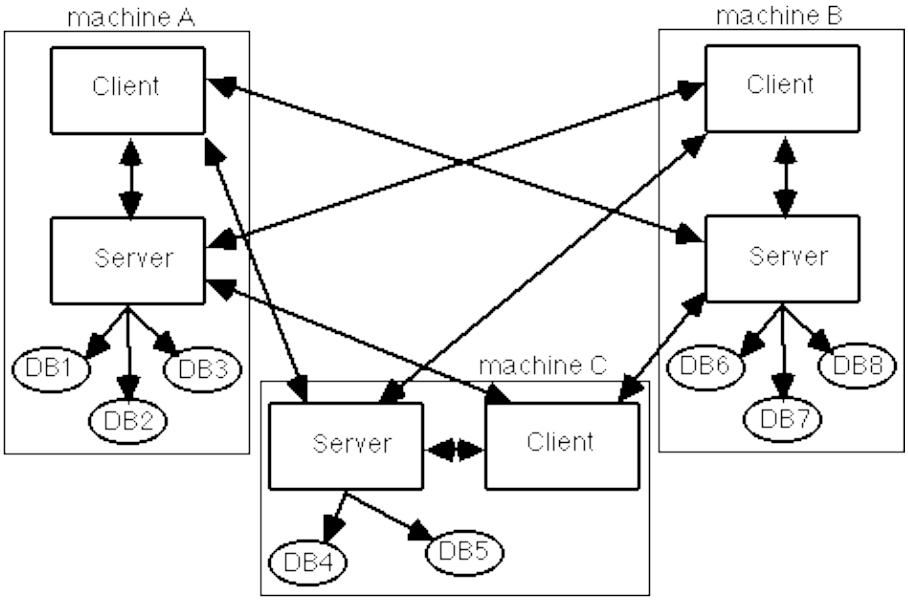
- **IPC overhead** For each transaction, the client must communicate the transaction to the server, adding to the overhead of each database access. This overhead is usually unimportant, since the time it takes for the client and server to communicate is small compared to the time it takes to read data from and write data to the database. If a central, super fast, networked server is available, it may actually be faster to ask it questions than it would be to access a database directly.

- **Administrative complexity** Some network administration is required, even in installations where the client and server will run on the same machine.

1.1. Configurations

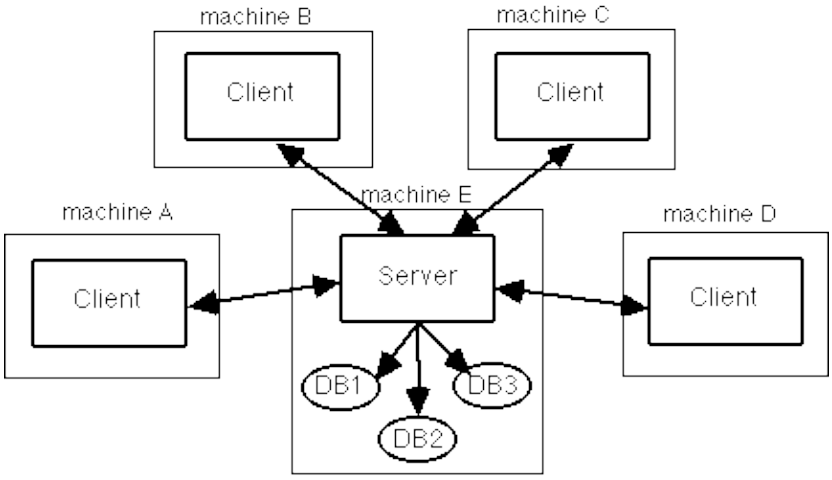
A wide variety of configurations are possible for THOR clients and servers. Two "extreme" configurations - fully distributed and fully centralized - are shown in the illustration below.

In the figure "Distributed Databases" (below), every machine contains one or more databases, and runs a server and one or more clients. Each client has access to all databases on all machines, thus forming a system in which the data and computing burden are spread across the whole network. This might be useful in a research laboratory in which a number of researchers independently collect data. Each researcher would be responsible for the data in one database, yet all researchers would have access to all data.



Distributed Databases

In the figure "Centralized Databases" (below), a central machine contains all databases and runs the server; all data are accessed by remote clients that connect to the central server. A scheme like this might be useful in a large corporation with a central database in which all corporate chemical information is stored. The central server would most likely be a large, expensive computer capable of handling large numbers of requests.



Centralized Databases

1.2. The Server

As mentioned earlier, a server is a relatively simple program. In computer lingo, it is a *daemon*- a process that runs in the background with no user interface of its own. Since it has no user interface (no terminal or window into which you can type commands), all communication to a server is done via client programs.

Each server keeps a log file of its activities. This file contains a record of certain selected client activities, such as client connect/disconnect events, security information, database openings and errors. The actual file's name is configurable; see the chapter on configuring and running servers for more details.

The following is a partial list of a server's responsibilities:

- Accept connections from clients; create a client context in which to store the names of databases the client has open and other client-specific data.
- Detect lost connections; close all databases that the client has open and discard the client context.
- "Ping" - echo a message back to a client (test a connection).
- Share database resources among clients (caching, open files, disk space, cpu). For example, a database that is opened by two clients doesn't use any more resources than one opened by a single client.
- Store and retrieve data.
- Rebuild databases that are damaged, "garbage collection".
- Provide information about databases (sizes, who is using it, etc.)
- Security. This is discussed in more detail below.

1.3 The Client

As we noted previously, a client is a more complex beast than a server. A server only has to deal with a well-specified set of requests from client programs. By contrast, client programs typically deal with users, who are unpredictable and demanding; client programs are correspondingly more complex.

Unlike a server, which is a single program supplied by Daylight, a THOR client is any program that uses the THOR Toolkit to access THOR Databases via a server. A client might be a program supplied by Daylight, one sold by a third party, or a program you write using the Daylight Toolkit. Whatever the origin of various clients, all are based on the Daylight Toolkit, the library of functions that allows programmers to write client

programs (applications). Clients can vary in complexity from simple to sublime.

In spite of the variety of clients that might be available, clients typically perform the following tasks:

- Connect to one or more servers.
- Open one or more databases.
- Retrieve and/or store information in databases.
- Ask servers for information about the servers and their databases.

1.4 Current Servers and Clients Applications

Servers	
Program	**Function**
thorserver	Record lookup; read/write database access; database management and creation.
merlinserver	Fast in-memory searching of database pools.

Clients		
Program	**Client of...**	**Function**
XVTHOR	thorserver	Read/write database access.
XVMerlin	thorserver and merlinserver	Exploratory data analysis of Merlin pools.
MCL	Merlinserver	Serial script-based Merlin client.
sthorman	THORserver and Merlinserver	THOR/Merlin server and database management.
thorchange **thorcrunch** **thordbinfo** **thordbping** **thordelete** **thordestroy** **thordiff** **thorlist** **thorload** **thorlookup** **thorls** **thormake** **thorping** **thorwho**	thorserver	Various serial programs for THOR management and general purpose database access.
merlindbping **merlinload** **merlinls** **merlinping** **merlinserver** **merlinsmartstalk** **merlinwho**	merlinserver	Various serial programs for Merlin management and general purpose database access.

2. THOR Server

THOR server must be started before any THOR programs can access a database. The server is a "daemon" program - it runs in the background and has no direct user interaction. Once you start a server, the only way to communicate with it is through a client program.

The servers write auditing information (e.g. the names of users as they connect and the databases each user opens) to a log file; the default is to use the standard-output file. Basic installation and configuration instructions for a single server can be found in the Daylight Installation Guide.

2.1 Starting Servers

We suggest starting the server with one of the following methods. Note that if your thorserver is killed unexpectedly, e.g. your system crashes, the thorserver will create lock files (database.LCK) for each database that was open when the server was killed. These files must be removed before restarting the thorserver.

Method 1 - Run server in an existing window

This method is useful when verifying that the server is installed correctly, as it allows you to see easily see the logged messages.

Choose the window you wish to use, and type:

```
$DY_ROOT/bin/thorserver
```

Once you have a server configured correctly, we suggest you use Method 2.

Method 2: Server in "background" with logfile

In this method, the server runs in "background" mode; its log-message output goes to a file. The log file can be monitored by one or more users via the `thorserv_monitor` program

Start the server in the "background" using a log file, type:

```
$DY_ROOT/bin/thorserver -THOR_LOG_FILE /tmp/thorserver.log &
```

You can monitor the server's log file using `thorserv_monitor` as follows:

```
$DY_ROOT/bin/thorserv_monitor /tmp/thorserver.log
```

If you start `thorserv_monitor` with no log file specified, it will use the log specified by the environment variable `DY_THOR_LOG_FILE`. You can monitor the server's log file from several locations simultaneously, or several users can monitor it. The `thorserv_monitor` program is passive, and doesn't affect the server in any way. For example, you might start the THOR server on a centralized server machine, then return to a computer in your office, from which you could "rlogin" to the server, set the `DISPLAY` environment variable (for X Windows) to your office-computer's display, and start a `thorserv_monitor` program. This would allow you to monitor the central server's activities from your office.

2.2 Stopping Servers

Killing a THOR server while users are logged in can be done but not recommended. The programs `sthorman` and `thorwho` tell you the names of users currently connected and which databases they are using. Alternately, you can examine the log file. Each server prints a list of all remaining connections when someone logs out. If the last line of the log file is "(none)", then it is safe to kill the server.

The programs `dayevict` and `sthorman` provide a means for "evicting" users, and optionally allow you to send a message to users and wait for a grace period; this gives users time in which to wrap up their business.

To kill a server started in a window (**Method 1** above), select the window the server is using, then type the interrupt character (usually [Control]-C but sometimes [Delete]). Alternatively, use `kill` with the process id the server printed on startup.

To kill servers started in the background (**Method 2** above), get the server's process id from the second line of the thorserver.log file and use it with the `kill` command.

Please note if you do kill a THOR server with databases still open, it will try to disconnect the clients and close the databases in a safe manner.

2.3 Auto-loading Databases

Specific databases can be opened at the time that the THOR server is started. For example, the command would start the THOR server and open the database 'database1' even when no clients are using it.

```
$DY_ROOT/bin/thorserver database
```

2.4 Multiple Servers

Normally a single instance of each server provides plenty of power for all users. However, it is sometimes necessary to run multiple THOR servers on a single computer. Please note that a database can be opened by only one server at a time and that you must have a license that allows multiple servers.

The following situations may call for multiple THOR servers:

- For different levels of security. You might have two completely separate sets of databases with different security requirements. An "open/public" server might be started using one userid, and a second "secure" server by another. Since each is run under a different user's id, each server would not be able to open the other's databases. Separate passwords files can be used (see the option DATABASE_PASSWORDS_FILE), so each server's security and access is independent of the other's.

- For exclusive access to a database. If you have a multi-user license, you can have it split across two or more services (e.g. a 7-user service and a 1-user service for an 8-user license). A THOR manager who connects to the 1-user thor server knows that no one else can connect, and thus knows that if he or she has a database open, no one else does. (Note: the "lock for exclusive access" feature is not implemented in version 4.41. Later versions of Daylight software may make it possible to lock a database without a one-user server.)

- When system limits are exceeded. The operating system imposes limits on each process; for example, many systems are configured to limit each process to 64 open files. In the case of THOR, the server starts with 4 open files, each database uses 4 files, and each client connection uses 1 file. If the 64-file limit is reached and it is not convenient to reconfigure the operating system, a second server can be a useful temporary solution.

To use multiple servers, each server must have a unique service in the `/etc/services` file. These THOR services must start with "thor" (e.g. "thor2", "thormgr", are permissible, "mthor" is not). We suggest something like following entries in /etc/services for two servers:

```
thor        5555/tcp
thor2       25557/tcp
```

To start a second server, use the `THOR_IPC_SERVICE` on the command line:

```
$DY_ROOT/bin/thorserver -THOR_IPC_SERVICE thor2
```

2.5 Sending Messages to Users

The programs `daymessage` and `sthorman` allow you to send messages to THOR client programs.

Messages can be "attached" to a THOR server, which causes the message to be sent to clients as they log in. They can also be attached to a database, in which case the message is sent to the client as it opens the database. Messages can also be "broadcast" immediately to all clients of a server, or to all clients who have a particular database open.

Each message is sent as one or more "NOTE:" entry in the normal error queue. It is up to the client program to examine the error queue and deal with the messages, so there is no guarantee that a particular client will actually deliver the message to a user. Daylight-supplied client programs (e.g. XVTHOR) attempt to post all messages as promptly as possible.

Messages should be short -- lines should be less than 70 characters wide, and the whole message should be just a few lines.

2.6 Monitoring and Maintenance

If your servers keep log files, you will occasionally (depending on activity levels) need to remove the files, as they will otherwise grow to take a significant amount of disk space. You can't simply remove a log file, since the server keeps it open for writing. (In UNIX, the space a file uses is not recovered until the last "reference" to it - a program or directory entry - is removed.) Instead, you have to stop the server, remove the file, then restart the server.

It is also a good idea to occasionally review the entries in the log file, in particular to check for repeated failed login attempts. This is often an indication that an unauthorized user is attempting to guess passwords or to otherwise violate security.

3. Merlin Server

Merlin servers must be started before any THOR or Merlin programs can access a database. The servers are "daemon" programs - they run in the background and have no direct user interaction. Once you start a server,

the only way to communicate with it is through a client program.

The servers write auditing information (e.g. the names of users as they connect and the databases each user opens) to a log file; the default is to use the standard-output file. Basic installation and configuration instructions for a single server can be found in the Daylight Installation Guide.

3.1 Starting Servers

We suggest starting the server with one of the following methods.

Method 1: Run server in an existing window.

This method is useful when verifying that the server is installed correctly, as it allows you to see easily see the logged messages.

Choose the window you wish to use, and type:

```
$DY_ROOT/bin/merlinserver
```

Once you have a server configured correctly, we suggest you use Method 2.

Method 2: Server in "background" with logfile

In this method, the server runs in "background" mode; its log-message output goes to a file. The log file can be monitored by one or more users via the merserv_monitor program).

Start the server in the "background" using a log file, type:

```
$DY_ROOT/bin/merlinserver -MERLIN_LOG_FILE /tmp/merlinserver.log &
```

You can monitor the server's log file using `merserv_monitor` as follows:

```
$DY_ROOT/bin/merserv_monitor/tmp/merlinserver.log
```

If you start `merserv_monitor` with no log file specified, it will use the log specified by the environment variable `DY_MERLIN_LOG_FILE`. You can monitor the server's log file from several locations simultaneously, or several users can monitor it. The `merserv_monitor` program is passive, and doesn't affect the server in any way. For example, you might start the Merlin server on a centralized server machine, then return to a computer in your office, from which you could "rlogin" to the server, set the `DISPLAY` environment variable (for X Windows) to your office-computer's display, and start a `merserv_monitor` program. This would allow you to monitor the central server's activities from your office.

3.2 Stopping Servers

Killing a Merlin server while users are logged in can be done but not recommended.

The programs `sthorman` and `merlinwho` tell you the names of users currently connected and which databases they are using. Alternately, you can examine the log file - each server prints a list of all remaining connections when someone logs out. If the last line of the log file is "(none)", then it is safe to kill the server.

The programs dayevict and sthorman provide a means for "evicting" users, and optionally allow you send a message to users and wait for a grace period; this gives users time in which to wrap up their business.

To kill a server started in a window (**Method 1** above), select the window the server is using, then type the interrupt character (usually [Control]-C but sometimes [Delete]). Alternatively, use `kill` with the process id the server printed on startup.

To kill servers started in the background (**Method 2** above), get the server's process id from the second line of the thorserver.log file and use it with the `kill` command.

Please note if you do kill a Merlin server with databases still open, it will try to disconnect the clients and close the databases in a safe manner.

3.3 Auto-loading databases

Specific databases can be opened at the time that the Merlin server is started. For example, the following command would start the Merlin server and load the database 'database2' into the server's memory. This makes the database available to client programs without the need to use sthorman (described below) to load the pool.

```
$DY_ROOT/bin/merlinserver database2
```

3.4 Multi-CPU Parallel Processing

If you have a multi-CPU machine, you can improve some search times dramatically by using several CPUs to "parallelize" certain Merlin searches. The environment variable `MERLIN_NPROCS` indicated to the merlinserver how many CPUs to use for searches. The default is one.

The actual parallelized search is performed by "child" processes, spawned by the merlinserver; for example, parallel substructure searching is handled by a program named merlinsmartstalk.

3.5 Multiple Servers

Normally a single instance of each server provides plenty of power for all users. However, it is sometimes necessary to run multiple Merlin servers on a single computer. Unlike the THOR server, it is possible for two Merlin servers to open the same database but it is not an efficient use of the computer's memory. Please note that you must have a license that allows multiple servers.

Reasons for using multiple servers might be:

- Merlin servers use a "time slicing" scheme to break lengthy operations into small sections, thus preventing one client from "hogging" the server. However, this time-slicing version of time sharing is not as sophisticated as the true time sharing employed by the UNIX operating system. In some cases, particularly for very large databases, performance can be improved by devoting a Merlin server to one particular database. This allows the operating system's more sophisticated time-sharing system to allocate search time to each client's requests.
- As with THOR, there may be separate security required for different databases.

To use multiple servers, each server must have a unique service in the `/etc/services` file. The Merlin services must start with "merlin" (e.g. "merlin2", and "merlinbig" are permissible but "bigmerlin" is not). We

suggest something like following entries in /etc/services for two servers:

```
merlin      5556/tcp
merlin2     5558/tcp
```

To start a second Merlin server, use the `MERLIN_IPC_SERVICE` on the command line:

```
$DY_ROOT/bin/merlinserver -MERLIN_IPC_SERVICE merlin2
```

3.6 Sending Messages to Users

The programs `daymessage` and `sthorman` allow you to send messages to Merlin client programs.

Messages can be "attached" to a Merlin server, which causes the message to be sent to clients as they log in. They can also be attached to a database, in which case the message is sent to the client as it opens the database. Messages can also be "broadcast" immediately to all clients of a server, or to all clients who have a particular database open.

You can "broadcast" messages to users of a Merlin database (pool), but Merlin can't reconfigure databases (it is a "read-only" server), so you must use the THOR server to attach a message to a database.

Each message is sent as one or more "NOTE:" entry in the normal error queue. It is up to the client program to examine the error queue and deal with the messages, so there is no guarantee that a particular client will actually deliver the message to a user. Daylight-supplied client programs (e.g. xvmerlin) attempt to post all messages as promptly as possible.

Messages should be short -- lines should be less than 70 characters wide, and the whole message should be just a few lines.

3.7 Monitoring and Maintenance

If your servers keep log files, you will occasionally (depending on activity levels) need to remove the files, as they will otherwise grow to take a significant amount of disk space. You can't simply remove a log file, since the server keeps it open for writing. (In UNIX, the space a file uses is not recovered until the last "reference" to it - a program or directory entry - is removed.) Instead, you have to stop the server, remove the file, then restart the server.

It is also a good idea to occasionally review the entries in the log file, in particular to check for repeated failed login attempts. This is often an indication that an unauthorized user is attempting to guess passwords or to otherwise violate security. This is discussed in more detail in the section *Server and Database Security*.

4. Server and Database Security

It can be difficult to secure a shared database against unauTHORized access without encumbering the users with a host of easily-forgotten passwords. This is especially true in a networked environment with a database server such as THOR: wide-area networks potentially give thousands of computers access to a THOR database, and security is a genuine problem.

There are *three distinct reasons* for security:

- To prevent unauTHORized access to data.

- To prevent auTHORized users from accidentally changing data.
- To keep a record of users who are accessing or changing data.

You should carefully consider these three needs separately, and keep the following three cardinal rules in mind when designing your security:

- Use secure passwords. Passwords should be at least 6 characters long, should not be in a dictionary, and should not be obvious (phone numbers, birth dates, project names, people's names, "buzzwords" in common use, etc.).
- Let people choose their own passwords. Don't assign passwords that mean nothing to the people who must remember them - the most common security violation occurs when users write down passwords because they can't remember them.
- Never make security cumbersome - it motivates people to find ways around it. You want to make it easy for people to do things right.

THOR provides two distinct security systems for server connections and for individual databases.

4.1 Security and the Operating System

Most operating systems, such as UNIX, provide file-access protection of some sort. Operating-system restrictions inevitably supersede the restrictions imposed by the THOR server, but because THOR databases are accessed by a server, all clients that connect to the server have the same operating-system permissions as the user who started the server.

Thus, in keeping with good security practices, the servers will refuse to run if started by a "powerful" user such as the super-user "root" in UNIX.

It is important that the passwords file itself be protected from tampering by unauthorized users. The standard installation procedure will, if followed, give a reasonable degree of security.

4.2 Server Security

The THOR and Merlin servers normally operate with a hosts/users security file that controls access to the server. Restrictions can be placed on individual users and on all users of particular host computers. The hosts/users file's location is specified by the environment variable `DATABASE_PASSWORDS_FILE`, typically set to `$DY_ROOT/etc/dy_passwords.dat`.

The passwords file defines the combination of hosts and users that can connect to the server. The user names in a server's host/users list are not taken from the operating system; rather they are maintained separately by the THOR/Merlin system. Although system administrators typically choose the same user names for the THOR/Merlin system as for the UNIX or VMS system (and client programs encourage this), there is no requirement that this be the case. The THOR/Merlin system uses a completely separate password validation system from that employed by the operating system (this makes the THOR/Merlin security system independent of the operating system on which it is running).

There are three levels of security available on the THOR and Merlin servers:

- No security: If the server is started with no passwords file, any user from any host can connect to the server. The server will log the transaction, but nothing more. If this configuration is used, the only protections are the database's read, write, and executive passwords. For example, a THOR server with

no security can be started as follows:

```
$DY_ROOT/bin/THORserver -DATABASE_PASSWORDS_FILE ""
```

- Equivalent Hosts: A group of computers can be considered equivalent hosts for security purposes; that is, you form a group of mutually-trustworthy computers (perhaps those in your building but not those on the rest of the campus) and list them in THOR's passwords file. Service is automatically granted to any user on an equivalent host without further validation (i.e. users don't have to supply a username and password if they are connecting from one of the equivalent hosts). Users who are not on an equivalent host must supply a valid username and password from the passwords file. The sTHORman program provides security-modification menu selections to select the equivalent-hosts mode.

  In this mode, individual user/password entries can also be added. Such users can log in from any computer, whether or not the computer is listed as an equivalent host.
- Restricted Hosts: Similar to equivalent hosts, but opposite in effect: Only users on the list of restricted hosts are allowed to connect to the server; in addition a user must supply a valid username and password when connecting to the server. Restricted-hosts-mode and equivalent-hosts-mode are mutually exclusive. sTHORman provides security-modification menu selections to select the restricted-hosts mode.

A typical passwords file might look like this:

```
host:crawdad
host:gator
host:cajun
user:art:1aA3h3azZaqw23DS
user:dawn:GjkO96REnK2G1Waw
user:jj:AsDF12REIO9PlLYb
user:thor:jfiIOe35W1QGh7cx
user:thorinfo:
```

In this example, any user from machines "crawdad", "gator", and "cajun" is allowed to log in and doesn't need to supply a password. Users "art", "dawn", and "jj" can connect from any computer if they supply a password that, when encrypted, matches the password for their usernames.

If the line `host:*only*` appears in the passwords file, then the hosts listed are considered *restricted hosts* as described above. If above line does not appear in the passwords file, then all hosts listed are considered *equivalent hosts*. Thus, if we added the line "`host:*only*`" to the above example, all users would be required to supply passwords, and only users on machines crawdad, gator, and cajun would be accepted.

It is not necessary to edit the THOR server's passwords file directly. The THOR manager program `sthorman` provides a way to change entries in the passwords file.

Please note that there are two special users listed in the password file. The user `THOR` is similar in concept to the "super user" in the UNIX system. Only the `THOR` password can be used to add or delete user and host entries from the passwords file, and the user `THOR` is allowed to change any other user's password.

The user `thorinfo` is is a restricted user that is allowed to connect and ask for only "non-critical" information such as if a particular database is loaded. This allows clients to ask a server questions of interest before having to enter a password. Note that `thorinfo` appears in the passwords file like any other user; you can remove it or add a password if you like. Of course, if you do so, the convenience features provided by this restricted user will not be available.

4.3 Database Security

Each database has three passwords, one each for read, read/write, and executive access. These passwords are encrypted and stored in an encrypted version. Database passwords can be set or reset using `sthorman` or `thorfilters`. If you have forgotten the executive password for a database, you have no recourse but to edit the `.THOR` file and remove the passwords, then use `sthorman` or `thorchange` to enter new ones.

5. THOR Database Administration

A THOR Database is a collection of chemical information, stored in a set of files on a computer's disk. This section discusses the practical aspects of managing THOR databases.

The tasks of a THOR administrator include the following:

- Database design. Deciding what is to be stored in it, how it is to be represented, who will have access to the data, where it will be located.
- Datatype design. Datatypes define the meaning of data in a THOR database; the THOR manager must design and document them.

- Creating and loading databases.
- Backups. Databases need to be dumped and stored on magnetic tape or some other removable medium on a regular basis.

There are two different techniques you can use for managing THOR databases:

`sthorman`
: An interactive, comprehensive THOR/Merlin management program.

`thorfilters`
: A collection of single-purpose UNIX-style "filter" programs. Each program does one task, such as create a database (`thormake`), load a database (`thorload`), list available databases (`thorls` and `merlinls`), etc.

Most THOR/Merlin administrators learn the basics of THOR databases management using `sthorman`. As certain tasks become rote, most administrators prefer to write short "shell scripts" using the thorfilters programs. Both `sthorman` and `thorfilters` are reasonable approaches to database management. You should choose whichever is suited to your needs.

Note: SGI IRIX 6.4 and higher and Solaris 5.7 and higher can use 34 bits to access files imposing a 16GB limit on new databases. Red Hat Linux 7.1 and higher and Mac OS X are limited to 32-bit access with a maximum database size of 2GB.

5.1 Database Creation and Maintenance

Databases can be created, loaded, dumped, crunched, reconfigured, and summarized using `sthorman` and/or the `thorfilters` programs. Remember when creating and maintaining databases that database paths (filenames and directories) are relative to the *server's* file system, whereas THOR Datatree files (TDTs) and `sthorman`'s log files are on the file system of the computer on which `sthorman` is running.

5.1.1 Creating and Loading a New Database

As THOR an administrator, you may have TDT files from which you would like to create a THOR database. Example TDT files are available in $DY_ROOT/data and $DY_ROOT/data/datatypes. Every database must have a "datatypes" and a "regular" TDT file. An "indirect" file is optional and only required if the "regular" database has indirect references.

When creating a database, you need the following information:

database name
: Name of the new database containing a complete path to the specific database files. For ease of maintenance, we recommend that you use environment variables such as `$DY_THORDB` to describe the path.

database type
: Whether this is a regular, datatypes, or indirect-data database.

datatypes database
: Name of the datatypes database in which the datatypes are defined.

indirect-data database
: Name of the indirect-data database in which indirect-data expansions are defined (only required if using indirect data)

primary database size
: The size of the primary-data hash table. This should be roughly equal to the number of TDTs that will ultimately be stored in the database. In general it is better to overestimate than underestimate since the hash table doesn't take much space (8 bytes per entry). See the size discussion below.

cross-reference database size
: The size of the cross-reference hash table. This should be roughly equal to the number of non-SMILES identifiers in the database. See the size discussion below.

read, write, and executive passwords
: The passwords for the database.

Before creating a new database, you must calculate how big to make the hash tables. The general rules is to make the hash tables the same size as the number of TDTs that are to be stored. This applies to both the primary data file and the cross-reference data file. The `tdtcount` program can be used to count bytes, identifiers, non-identifiers, SMILES, and total number of TDTs in a file.

Example:

```
tdtcount example_datatypes.tdt

9562 32 231 0 32 example_datatypes.tdt
tdtcount example.tdt
```

270398 1601 3333 232 232 example.tdt For this example we have 32 TDTs with one identifier per TDT (32 total) for the datatypes file. In addition, we have 232 TDTs with 1601 identifiers for the regular file. In order to create and load a new database you need to follow the follow steps using either `thorfilters` or `sthorman`. Note that the indirect-data steps are only required if using indirect data.

- Create the datatypes database
- Create the indirect-data database
- Create the regular database
- Load the datatypes database
- Load the regular database
- Load the indirect database

Example using `thorfilters` and the results from the `tdtcount` performed above:

```
thormake '$DY_THORDB/example_datatypes' 32 32
thormake -DATATYPES_DATABASE example_datatypes '$DY_THORDB/example' 232
1601
thorload example_datatypes example_datatypes.tdt
thorload example example.tdt
```

5.1.2 Loading New Data

New data can be loaded into an existing database using `sthorman` and/or `the thorfilters` programs. Before loading new data into a THOR database, you need to know the following information:

TDT file
The name of the data file that is to be loaded.

database
The database into which the file is to be loaded.

password
You need the write password to load new data.

error file
You can specify a file into which error messages will be reported or let error messages come to the screen.

merge data
Normally, data being loaded should be *merged* with the data already in the database - that is, if a TDT is entered that is already present in the database, the new and old data are merged into a single TDT. This is true even when the database is initially empty, as a particular SMILES or other identifier may occur multiple times in the input file.

replace existing data
If you choose *not* to merge new data with existing data, there are two possible choices: the new data should *replace* existing data, or new data that conflicts with existing data should *not be loaded*.

Note that replacing existing data requires that the database be opened with *executive* permission.

standardize/raw
You can choose to *standardize* (see the Daylight Theory Manual) new data or to load the data *raw*. Normally you should *standardize all data* that are loaded into a THOR database. The *raw* option should only be used when re-loading data that is already standardized (i.e. data from a previous dump of a THOR database). Loading *raw* data is much faster than standardizing, but you must be positive that the TDTs are already standardized.

Note that datatype-definition TDTs (those being loaded into a datatypes database) cannot be merged. When re-loading datatype definitions, you must replace the existing definitions.

major/minor report frequency
As TDTs are loaded, `sthorman` and `thorload` indicate progress by printing dots ("." - minor report) and a numeric summary (major report) on the terminal. You can choose how often these reports are made (i.e. how many TDTs between reports).

5.1.3 Dumping a Database

There are three ways to dump a database's contents using: `sthorman`, `thorlist` or `thordump`. Before you can dump a database, you need the following information:

name
The name of the database to be dumped.

password
The read-access password for the database, if any.

TDT file
The file into which you want to store the THOR datatrees.

expand indirect
Whether you want to expand indirect references or not. If you expand them, the original indirect-reference identifiers are lost, but the TDTs are suitable for printing or other uses where people will read them. Normally, indirect references are not expanded when archiving a database; instead, the indirect-data database is archived separately.

major/minor report
How many TDTs between the "dots" in the "progress report". While the database is being dumped the programs print dots, by default every 100 TDTs, and a brief summary every 5000 TDTs. You can change these values.
5.1.4 File Ownership and Privileges

All of the files in that constitute a THOR database should be owned by the THOR manager (i.e. the user who will start the THOR server), and must have read *and* write permission for the THOR manager (this is true even if the database is only opened with read permission). No other users should have read or write permission to the THOR database files and directories.

All directories in the database's path must be readable by the THOR server, and the THOR server must have permission to change to each directory in the path. If a path specified to open a database contains symbolic links, logical symbols, or other ways that generate "aliases" for the real path, the real path and the "alias" path must be readable by the THOR server.

5.1.5 Moving databases

It will occasionally be necessary to move a database, e.g., from a nearly-full disk to a new or larger disk. If you are moving all of DY_THORDB (that is, you are redefining DY_THORDB), then perform the following steps:

- Kill the servers

- Move the databases
- Redefine DY_THORDB
- Restart the servers

If you are moving a database but *not* restarting the server or redefining DY_THORDB, it is necessary to update the the database's "header" file, the one with the suffix `.THOR` with the new location of the other files that make up the database (indirect and datatypes). Alternatively, you might be to define an environment variable `DY_THORDB2` to be the new location. Then, if we ever need to move the database again, we can do it by redefining `DY_THORDB2` rather than editing the database header file.

5.1.6 Using Multiple Disks

For installations with large databases, it may be necessary to store databases on two or more disks. The recommended method for this is to define several environment variables instead of the usual DY_THORDB. For example, one might define the following before starting the THOR and Merlin servers (sh(1) syntax is shown):

```
DY_THORDB1=/thordb
DY_THORDB2=/bigdisk
DY_THORDB3=/hugedisk
DATABASE_PATH="$DY_THORDB1 $DY_THORDB2 $DY_THORDB3"
export DY_THORDB1 DY_THORDB2 DY_THORDB3 DATABASE_PATH
```

In extreme situations, it is possible to store parts of the same database on more than one disk. For example, the primary data file (.DP) and the cross-reference data file (.DX) usually use the most space; they could be moved to different disks, and the database header file (.THOR) edited accordingly.

5.1.7 Backing Up and Restoring Data

There are several ways to back up a THOR database: the `sthorman` database-dump capability, `thorlist` to generate a TDT file, simply copy and `tar` the actual database files (not recommended), or `thordump` as a "last resort".

5.2 Record Locking

Record locking can be specified for a THOR database: when record locking is in effect, a client program can lock a TDT for exclusive write access. While a TDT is locked, the "owner" of the lock (the client that set the lock) is the only client who can modify the TDT. Other clients can retrieve the TDT "read only", but attempts to retrieve a writable TDT or to lock the TDT will fail.

Changes made to a locked record are invisible to other clients until that record is unlocked ("committed"). For example, if a client locks a record, then deletes it, other clients will still "see" the original record, but to the client holding the lock the record will appear to be gone. It is possible undo changes made to a locked record (do a "rollback") at any time before it is unlocked.

Daylight-supplied client programs, such as `thorload` and `sthorman` don't handle TDT locking in any specific way. They just go about their business, and if they attempt to modify a locked record, they report the error that results.

Record locking is mostly of interest to those who are writing their own programs using the THOR Toolkit. See in particular the Toolkit functions: `dt_thor_settdtlocking`, `dt_thor_tdtlocking`, `dt_thor_tdtget`, and `dt_thor_tdtput`.

You can enable and disable TDT record locking with the programs thormake, thorchange, or sthorman.

5.3 Read-Only Databases

Normally, THOR databases can be opened with read, write, or executive permission (assuming you know the correct password). But you can specify that a database be *read-only*, thus prohibiting write and executive operations on the database.

When a THOR database is *not* marked *read-only*, the THOR server opens the files using write permission, even if a client only requests that it be opened read-only. The reason is that several clients can have a database open simultaneously, but the THOR server only opens the files once, sharing the database among clients. Thus, THOR has to be prepared to carry out read, write and executive operations on a database even though a client only requests read permission.

However, if a database is marked *read-only*, the THOR server knows that no write or executive operations can be used, so it uses read permission to open the database files. Furthermore, the THOR server doesn't set a lockfile on a read-only database. (This also means that multiple THOR servers can open a read-only database, something that is otherwise prohibited.)

Databases that are to go on a read-only medium, such as a CD-ROM or a read-only file system, must have their *read-only* property set. You can set and unset the *read-only* property with the programs thormake, thorchange, or sthorman.

6. Merlin Pool Administration

Merlin administration administrative tasks are:

- Computing available memory and designing datatypes accordingly.
- Loading databases into memory (at which point they are called "pools").

A Merlin pool is a subset of a THOR database: The administrator selects which datatypes are to be loaded via the _P<> datatype-definition dataitem. Datatypes are described in the Daylight Theory Manual.

6.1 Memory usage

It is critical that merlinserver have enough memory to load all active databases *without swapping*. The basic principle on which Merlin is based is that memory is many orders of magnitude faster than disk. If you try to load more data than will fit into memory, Merlin's performance will degrade to unusable. Thus, it is critical that you design your datatype definitions to only load as much data as will fit.

It is also possible with Merlin to use up all of your computer's swap space. The design of UNIX makes it impossible for any program (such as merlinserver) to give allocated memory back to the operating system. Thus, if you attempt to load more data into a Merlin server than will fit in your system's swap space, the Merlin server will report that it was unable to load the pool, *and* it will have used all of your swap space. At this point you won't be able to run any other processes. To prevent this, you can configure the MERLIN_MEMORY_LIMIT. See Daylight Installation Guide for more information.

Note: SGI IRIX 6.4 and higher and Solaris 5.7 and higher can use 64 bits to access memory. The 64-bit Merlin server (named $DY_ROOT/bin64/merlinserver) can access more memory than can be assembled on a

conventional computer, making the pool size virtually unlimited. The 32-bit Merlin server (named $DY_ROOT/bin/merlinserver) uses 32 bits to access memory and is limited to about 1.7GB. Red Hat Linux 7.1 and higher and Mac OS X uses 32 bits to access memory and is limited to about 2GB.

6.2 Loading a Merlin Pool

There are three ways to load a database into the Merlin server's memory using: `sthorman`, `merlinload`, or `merlinserver` when the server is started.

Loading a database into the Merlin server's memory can take a significant amount of time, since the entire database must be scanned. Database loading is a "monolithic" operation - while a database is loading, the server will not respond to any client requests. Note: When a database is released, the server will not actually remove it from its memory until the last client has closed it.

6.3 Columns and Cells

Although a pool typically contains a subset of the data in a THOR database, it often consists of a wide variety of types of information, including structure, reactivity, chemical properties, prices, catalog numbers, and so forth. At any particular moment, the typical user is only interested in a few types of data. To solve this problem, Merlin provides *columns*.

A column of data is conceptually a "vertical slice" through all datatrees; it selects a specific datum or *derived-datum* (the result of computations on real data) from each datatree. The particular datum used for a column is defined by two properties:

Datatype:
When creating a column, you specify the datatype and the field within that datatype.

Function:
A datatree may contain several of a particular datatype (for example, there may be many names). Several *functions* are available to select from among those available, including "first", "last", "average" (for numeric data), "least", "greatest", and so forth; these are explained in detail below.

The pool can be thought of as a "chemical spreadsheet," with one row for each datatree in the database, and various columns for different types of data. The intersection of a row and a column is called a *cell*, and is the basic unit, or datum, in Merlin.

A *derived-data column* is special type of column whose data are not in the database, but are derived from the database or computed during operations on the database. For example, a SIMILARITY column contains data computed when you perform a structural-similarity search. There is no SIMILARITY data in the database itself; the data are created and changed as you work. Other types of derived-data columns are discussed in the section below.

6.4 Column-creation functions

As discussed above, *column-creation functions*, or simply *functions*, allow you to specify which instance of a particular datatype to use when more than one occurs in a row of the pool. The functions are:

First
Use the first instance of the datafield in the row.

Last
Use the last instance of the datafield in the row.

Least
Use the lowest-valued instance of the datafield. For ASCII data, this is the lowest lexical value; for numeric data it is the lowest numeric value.

Greatest Like **least**,
but use the greatest-valued instance.

Longest
Use the datafield that is the longest (contains the most characters). Not applicable to numeric data.

Shortest Like **longest**,
but use the shortest.

Count
Creates a derived-data column containing the number of instances of the datafield.

Average
Creates a derived-data column containing the average of all instances of the datafield. Only applies to numeric datatypes.

Standard Deviation
Creates a derived-data column by computing the standard deviation of all instances of the specified datafield. Only applies to numeric datatypes.

6.5 Loading Pools from Alternate Sources

Beginning with version 4.94, Merlinserver can load pools from alternate data sources using a program-object-based TDT interface. Basically any program object which outputs TDTs can be used as a data source for a Merlin pool.

The special 'datasource:' tag in a database .THOR file, if present, causes the merlinserver to execute the command after the 'datasource:' tag as a program object and to load the returned TDTs from the program object into the pool. The 'datasource:' tag can be used for regular, indirect, and datatypes databases. When the 'datasource:' tag is present, merlinserver will ignore the 'hash file' and 'data file' entries and will ONLY load from the program object.

A THOR file with a 'datasource' tag can not be read by thorserver. When using the 'datasource' tag one can not find the TDT in Thor that corresponds to a row in Merlinserver.

7. sthorman

`sthorman` (serial THOR manager - `$DY_ROOT/bin/sthorman`) is a tool used to manage the THOR/Merlin system. It is a menu-driven client program for serial terminals, and provides a simple user interface for use on all serial terminals.

`sthorman` provides full and integrated access to almost all THOR's and Merlin's database management capabilities. Using `sthorman`, you can reconfigure servers, build and maintain THOR databases, load Merlin pools, ask the server who is connected and what databases they are using, add and modify Daylight

users and passwords, specify caching parameters, summarize database contents, and much more.

When sthorman starts, it prompts you for the name of the server to which it is to connect and then the service (default of "thor"). You can (via the top-level menu) switch to another server at any time. The current server is the only *context* that sthorman keeps from one command to the next. Note that while some THOR and Merlin client programs will connect to multiple servers simultaneously, sthorman is single-minded and is always connect to exactly one server.

7.1 Command-line Options

sthorman has several command-line options that allow you to control its behavior.

-host *machine*
: Specifies the hostname of the machine on which the server is running. The default is the hostname of the machine on which sthorman is running.

-service *name*
: Indicates the IPC service (see /etc/services) that sthorman is to use; the default is thor. The service-plus-hostname combination constitutes a complete name for a server.

-script *filename*
: Names a file that will receive a transcript of all activities, in a form that can be used as a "script file" by the -input option, below. While producing a script file, sthorman's behavior is somewhat altered: It will not offer default choices, but rather requires that all questions be answered explicitly. This prevents the user from choosing defaults that might change from run to run. See the section *Batch processing* below.

-input *filename*
: Indicates that sthorman is to take its input from *filename* rather than from the keyboard. *filename* is generally a "script" file produced by a previous session of sthorman via the -script option above.

-quiet
: Used when sthorman is running a script (or "batch") file using the -input option, above; indicates that sthorman is to discard normal output rather than printing it on standard output.

-scroll *integer*
: By default, sthorman does not page standard output. If text rolls off the top of the display before it can be read, the scrolling region size should be specified with this option.

-help
: Print a synopsis of these options and quit.

7.2 Daylight Options

sthorman reads several options from the Daylight options manager, and therefore makes use of the environment variable $DY_ROOT. The following options affect sthorman:

THOR_IPC_SERVICE
: Controls the default answer that sthorman will provide when it asks you what server to connect to.

STHORMAN_STRINGS_FILE
: sthorman's menus, help messages, and error messages are taken from an ASCII text file. The factory default for this option is $DY_ROOT/etc/sthorman_strings.dat, which is the most

"verbose" of the versions.

7.3 Navigating

`sthorman` is a menu-driven program. Each menu is a numbered list of choices; at the prompt you select one of the choices by entering its number. Below is an illustration of starting `sthorman` (connected to server "dia" and service "thor") and asking for the version:

```
%sthorman
Version: sthorman 4.91

          ..................................................
          .  STHORMAN -- Serial THOR Manager Program.       .
          .                                                 .
          .  Enter "?" for help at any prompt.              .
          .  The menu selection "." goes to top-level menu. .
          ..................................................

STHORMAN needs to be connected to a server.

Enter machine the server is running on: ("dia")
Enter port (service) of server: ("thor")
Enter username: ("thor")
Enter server username's password:

Connected to new server.
Current service:  dia:thor

SERVER MENU (TOP LEVEL MENU):
1) Connect to a new server
2) List server's version
3) List current users
4) Change server security...
5) Databases...
6) Send message to users...
7) Evict user...
0) Quit
Enter menu selection (or '?'): 2

   Current service:  dia:thor
thorserver 4.91)
```

When using `sthorman`'s menus, the following should be kept in mind:

- A menu item is selected by entering its number.
- Some menu entries (e.g. "List server's version" in the above example) perform actions, while some take you to other menus.
- Menus are like a "tree": menu item "0" (zero) always selects the next menu "up" (back where you came from).
- The special menu selection "." (a period) always takes you back to the top-most ("SERVER") menu.
- An end-of-file (typically [Ctl]-D on your keyboard) always causes sthorman to quit immediately.

When `sthorman` asks you a question, it often provides a default answer. If you simply press the [Return] (or [Enter]) key, it is as though you typed the default.

Context-sensitive help is available virtually everywhere in `sthorman`. Whenever `sthorman` is waiting for your input, typing a "?" (question mark) will elicit a help message appropriate to the situation.

7.4 Functionality

`sthorman`'s top-level menu (the "SERVER" menu) provides choices to display information on the current server: who is connected, what databases they have open, the permission (read/write/executive) used to open the databases, and the current server's software level. Note that these menu items apply to both THOR and Merlin servers.

THOR and Merlin servers maintain their own user/passwords file, separate from that of the operating system (This is discussed at length in the chapter on *Server and Database Security*). The "SERVER SECURITY" menu allows you to inspect the server's security system, add new users and hosts, change passwords, and remove users and hosts. Note that the server-security menu items apply to both THOR and Merlin servers.

The SERVER SECURITY MENU appears as follows:

```
SERVER SECURITY MENU:
1) List users
2) List hosts
3) Add/Change user
4) Add host
5) Remove user
6) Remove host
7) Set exclusion mode
0) Go back to server menu
Enter menu selection (or '?'):
```

The "DATABASE" menu provides all functions needed to build and maintain THOR databases, and to load, release, and otherwise manage Merlin pools.

THOR and Merlin:

Database-search path
: The servers maintain a *search path* - a list of directories in which databases are to be found. When a client asks to open a database, if that database's name is a *relative name* (that is, it only names the database (file), not the directory in which the database resides), the server checks all the directories in its search path for the specified database. This makes it more convenient for users to specify databases, since they don't need to specify the full file-system path to the database.

Database security
: Each database is protected by three passwords, for read, read-write, and executive access. These passwords can be changed through either a THOR or a Merlin server. Security issues are discussed at length in the *Security* chapter, below.

Database information
: Both THOR and Merlin will provide information about databases: list all "known" databases (those in the search path), show whether anyone is using a particular database, and so on.

THOR only:

The THOR DATABASE MENU appears as follows:

```
DATABASE MENU:
 1) List known databases
 2) Change database search-path...
 3) Get database information...
 4) Create   a database...
 5) Load     a database
 6) Dump     a database
 7) Crunch   a database
 8) Change   a database's configuration...
 9) Destroy  a database
10) Send message to users of a database...
11) Evict users of a database...
 0) Go back to server menu
Enter menu selection (or '?'):
```

Get database information...
: Report header/configuration information or a content summary of the database.

Create database
: Create a new database, set its datatypes and indirect databases (if any).

Load database
: Load a TDT file into a database

Dump database
: Dump the contents of a database to a TDT file

Crunch database
: Recover unused space in a database.

Change database
: Change a database's auxiliary databases, passwords, crunch level, cache level, read-only status, record-locking status, or hold a database.

Destroy database
: Delete database files from disk.

Merlin Only:

The Merlin POOL MENU appears as follows:

```
POOL MENU:
1) List known pools
2) Change database search-path...
3) Pool: Load database and hold in memory
4) Pool: Release from memory
5) Column: Create and hold in memory
6) Column: Release from memory
7) Show pool info
8) Evict users of a pool...
0) Go back to server menu
Enter menu selection (or '?'):
```

Pool: Load database and hold in memory
: Load a database pool into the Merlin server's memory.

Pool: Release from memory
: Release a pool from the Merlin server's memory.

Column: Create and hold in memory
: Create a columns for searching a loaded datafield. This will save startup time for clients wishing to search this datafield.

Column: Release from memory

Release a column previously created.

Show pool info
Report status of pool.

Evict users of a pool
Disconnect users from a pool.

7.5 Batch Processing

`sthorman` has three options that control input and output, making it possible to operate `sthorman` in "batch" mode:

`-script` ***file***
With this option, `sthorman` writes a *script file* - a recording of everything you type and of `sthorman`'s responses (which are recorded as comments so that the file can be used with the `-input` option).

`-input` ***file***
With this option, `sthorman` reads from *file* rather from standard input. Input is usually a script file made by `sthorman` with the `-script` option, above.

`-quiet`
With this option, `sthorman` does not print anything on standard output. This is typically used with the `-input` option, above. Without `-quiet`, `sthorman` prints all questions and answers during "batch" operation with the `-input` operation. Note that the `-quiet` option should not be used in interactive situations, as it makes the interaction somewhat one-sided.

The script files produced by the `-script` option have several special features:

- All of sthorman's responses are "commented out" so that the file can be used as an input file.
- All passwords you enter are encrypted so that the original password isn't readable. The password can only be decrypted by sthorman (see the following subsection).

In addition, the `-input` specification causes `sthorman` to not offer default answers that might change from one invocation to the next. For example, in normal interactive mode, `sthorman` will present you with a menu of "known databases" for various database operations; you simply choose from this menu. But with the `-input` option, you are forced to specify the choice "other" from the menu, then explicitly enter the database's name. This is necessary because the database-selection menu changes as databases are added and/or deleted from the server's search path, and as the search path is changed. A particular choice on the menu will have a different meaning at a later invocation of `sthorman`.

When `sthorman` is run to make a script file for batch processing, actual `sthorman` operations are executed. This works fine if the reason for running batch is a repetitive task (e.g. checking connected users or backing up databases). If the batch task is to be done only once (e.g. building a large database over the weekend), a script file should be made with different specifications (e.g. building a smaller test database); and then edited using a standard editor such as vi(1).

It is not appropriate to store passwords in "plain text" in a file. If a file contains plain-text passwords, any user who has or can get read-access to your files can discover the password. Furthermore, encryption by itself is not enough, since even a encrypted password could be used by anyone with access to `sthorman`.

To prevent such breaches of security, `sthorman` encrypts all passwords before storing them in a batch file, and decrypts them when reading from a batch file. Passwords are encrypted in such a way that the user/machine becomes a decryption key; only the same machine and the same user can use a batch file that

contains encrypted passwords.

This means that no special precautions need to be used with `sthorman` script files that contain passwords. The passwords can't be discerned by reading the script file, and are not useful to anyone except the person who created them.

8. thorfilters

"THORfilters" is a general name for a set of programs that perform THOR/Merlin database-management tasks. Their capabilities are similar to those provided by `sthorman`, however, some tasks are better done with `sthorman` and some with `thorfilter` programs.

In particular, `thorfilter` programs can be used in custom scripts for database management tasks, such as building databases, reloading pools, archiving databases, and extracting database subsets. Another possibility would be using THORfilters such as thorlookup via CGI scripts to provide the back end to a Daylight database web-interface.

The name `thorfilter` is derived from the UNIX tradition of "filter" programs (such as sed, grep and sort), each of which performs a single text-processing task. UNIX filter programs are treated as "building blocks", and are typically "piped together" to perform complex tasks.

This building-block approach is the basis for thorfilters. Each of these programs performs one task, such as creating a database (`thormake`), loading data into a database (`thorload`), or listing available databases (`thorls`). By combining these building blocks, you can perform complex THOR- and Merlin-database management tasks.

8.1 THOR Programs

tdtcat	Read and write THOR Data Trees (TDTs) in either list (one data item per line) or dump format (one datatree per line)
thorchange	Change the properties of a THOR database
tdtcount	Count bytes, identifiers, non-identifiers, and TDTs
thorcrunch	Recover unused space in a THOR database
thordbinfo	Report the configuration of a THOR database
thordbping	Verify the existence of a THOR database
thordestroy	Destroy a THOR database (permanently erase it)
thordelete	Delete TDTs from a THOR database
thordiff	Compare the contents two THOR databases
thordump	Directly dump the contents of a THOR database from the primary datafile
thorlist	Dump the contents of a THOR database to a TDT file
thorload	Load data into a THOR database
thorlookup	Look up a TDT in a THOR database
thorls	Ask a THOR server to list its databases
thormake	Create a THOR database and/or set its properties
thorping	Verify the existence of a THOR server
thorwho	Ask a THOR server to list current users

8.2 Merlin Programs

merlindbping	Verify the existence of a Merlin database
merlinload	Load a database into the Merlin server's memory
merlinls	Ask a Merlin server to list its databases
merlinping	Verify the existence of a Merlin server
merlinwho	Ask a Merlin server to list current users

8.3 Generic Programs

dayevict	Evict clients from THOR/Merlin server or database
daymessage	Attache a message to THOR/Merlin servers or THOR database to be sent at client login or "broadcast" a message immediately to all clients

8.4 Entering Database and Server Names

All thorfilter programs (and most other Daylight programs that use THOR and Merlin databases) use a single syntax for naming servers and databases.

Briefly, it is like this:

database%*dbpass*@*host*:*service*:*user*%*userpass*

Where:

database	name of the database being opened
dbpass	password for the database
host	computer on which the server is running
service	IPC service (or "port") the server is using
user	user name for logging into server
userpass	password for "user" for logging into server

Most of the fields in the specification are optional; defaults are automatically supplied for those that you leave out. Note: This syntax is discussed in more detail in the database and server man pages.

If the database's name does not include passwords for the server and/or the database, the thorfilter program you are using will prompt you for them. If a database and/or server doesn't require a password, you indicate this by putting in the % with no password after it. For example:

mydb@zeus	You are asked for the database and server passwords
mydb%@zeus%	Use blank password for database and server
mydb%@zeus	Use blank password for database, ask for server password

Note that you will be prompted for passwords even if you redirect input and output - the thorfilters programs use a special file to prompt you that goes to your terminal even when I/O is redirected to pipes or files. (See the option SECURE_PASSWORDS, below.)

8.5 Generic Options

There are several options that are common to all thorfilters programs. We describe them once here, rather than in each thorfilter program's description.

SECURE_PASSWORDS TRUE|FALSE
: Normally, if passwords are entered on the command line (e.g. `mydb%mypass`), a thorfilter program will exit immediately. This is to discourage password discovery via programs, such as `ps(1)`, that show the complete command line used to invoke a program. If `SECURE_PASSWORDS` is set to `FALSE`, passwords can be entered on the command line. See the Help Widget's "database_name" topic for more details.

THOR_IPC_SERVICE service
MERLIN_IPC_SERVICE service
: Names the default service (see `/etc/services`) or "port" of the THOR server or Merlin server, respectively. Defaults are `thor` and `merlin` for THOR and Merlin programs, respectively. Note that these options are ignored if a server's name includes the service (e.g. `@crawdad:thor2`).