Chapter 16. HyperSQL on UNIX

How to quickly get a HyperSQL Listener up and running on UNIX, including Mac OS X

Blaine Simpson

The HSQL Development Group

$Revision: 6683 $

2024-06-01

Table of Contents

Purpose
Installation
Setting up Database Catalog and Listener
Accessing your Database
Create additional Accounts
Shutdown
Running Hsqldb as a System Daemon

- Portability of `hsqldb` init script
- Init script Setup Procedure
- Troubleshooting the Init Script

Upgrading

Purpose

This chapter explains how to quickly install, run, and use a HyperSQL Listener (aka Server) on UNIX.

Note that, unlike a traditional database server, there are many use cases where it makes sense to run HyperSQL without any listener. This type of setup is called *in-process*, and is not covered here, since there is no UNIX-specific setup in that case.

I intend to cover what I think is the most common UNIX setup: To run a multi-user, externally-accessible catalog with permanent data persistence. (By the latter I mean that data is stored to disk so that the catalog data will persist across process shutdowns and startups). I also cover how to run the Listener as a system daemon.

When I give sample shell commands below, I use commands which will work in Bourne-compatible shells, including Bash and Korn. Users who insist on using the inferior C-shells will need to convert.

Installation

Go to http://sourceforge.net/projects/hsqldb and click on the "files" link. You want the current version. I can't be more specific because SourceForge/Geeknet are likely to continue changing their interface. See if there's a distribution for the current HSQLDB version in the format that you want.

If you want a binary package and we either don't provide it, or you prefer somebody else's build, you should still find out the current version of HyperSQL available at SourceForge. It's very likely that you can find a binary package for your UNIX variant with your OS distributor, http://www.jpackage.org/, http://sunfreeware.com/, etc. Nowadays, most UNIXes have software package management systems which check Internet repositories. Just search the repositories for "hsqldb" and "hypersql". The challenge is to find an *up-to-date* package. You will get better features and support if you work with the current stable release of HyperSQL. (In particular, HyperSQL version 2.0.0 added tons of new features). Pay attention to what JVM versions your binary package supports. Our builds (version 2.0 and later) document the Java version it was built with in the file `doc/index.html`, but you can't depend on this if somebody else assembled your distribution. Java jar files are generally compatible with the same or greater major versions. For example,if your `hsqldb.jar` was built with Java 6, then it is compatible with Java versions 6 and greater.

It could very well happen that some of the file formats which I discuss below are not in fact offered. If so, then we have not gotten around to building them.

Binary installation depends on the package format that you downloaded.

Installing from a .pkg.Z file	This package is only for use by a Solaris super-user. It's a System V package. Download then uncompress the package with uncompress or gunzip
	
	`uncompress filename.pkg.Z`
	
	You can read about the package by running
	
	`pkginfo -l -d filename.pkg`
	
	Run pkgadd as root to install.
	
	`pkgadd -d filename.pkg`

Installing from a BSD Port or Package	You're on your own. I find everything much easier when I install software to BSD without their package management systems.

Installing from a .rpm file	Just skip this section if you know how to install an RPM. If you found the RPM using a software management system, then just have it install it. The remainder of item explains a generic command-line method which should work with any Linux variant. After you download the rpm, you can read about it by running
	
	`rpm -qip /path/to/file.rpm`
	
	Rpms can be installed or upgraded by running
	
	`rpm -Uvh /path/to/file.rpm`
	
	as root. Suse users may want to keep Yast aware of installed packages by running rpm through Yast: `yast2 -i /path/to/file.rpm`.

Installing from a .zip file	Extract the zip file in an ancestor directory of the new HSQLDB home. You don't need to create the `HSQLDB_HOME` directory because the extraction will create a version-labelled directory, and the subdirectory "hsqldb". This "hsqldb" directory is your `HSQLDB_HOME`, and you can move it to wherever you wish. If you will be upgrading or maintaining multiple versions of HyperSQL, you will want to retain the version number in the directory tree somehow.
	
	`cd ancestor/of/new/hsqldb/home`
	`unzip /path/to/file.zip`
	
	All the files in the zip archive will be extracted to underneath a new subdirectory named like `hsqldb-2.7.3/hsqldb`.

Take a look at the files you installed. (Under `hsqldb` for zip file installations. Otherwise, use the utilities for your packaging system). The most important file of the HyperSQL system is `hsqldb.jar`, which resides in the subdirectory `lib`. Depending on who built your distribution, your file name may have a version label in it, like `hsqldb-2.7.3.jar`.

Important

For the purposes of this chapter, I define `HSQLDB_HOME` to be the parent directory of the lib directory that contains `hsqldb.jar`. E.g., if your path to `hsqldb.jar` is `/a/b/hsqldb/lib/hsqldb.jar`, then your `HSQLDB_HOME` is `/a/b/hsqldb`.

Furthermore, unless I state otherwise, all local file paths that I give are relative to the `HSQLDB_HOME`.

If the description of your distribution says that the `hsqldb.jar` file will work for your Java version, then you are finished with installation. Otherwise you need to build a new `hsqldb.jar` file.

If you followed the instructions above and you still don't know what Java version your `hsqldb.jar` supports, then try reading documentation files like `readme.txt`, `README.TXT`, `INSTALL.txt` etc. (As I said above, our newer distributions always document the Java version for the build, in the file `doc/index.html`). If that still doesn't help, then you can just try your `hsqldb.jar` and see if it works, or build your own.

To use the supplied `hsqldb.jar`, just skip to the next section of this document. Otherwise build a new `hsqldb.jar`.

Procedure 16.1. Building hsqldb.jar

1. If you don't already have Ant, download the latest stable binary version from http://ant.apache.org. cd to where you want Ant to live, and extract from the archive with

   ```
   unzip /path/to/file.zip
   ```

   or

   ```
   tar -xzf /path/to/file.tar.gz
   ```

   or

   ```
   bunzip2 -c /path/to/file.tar.bz2 | tar -xzf -
   ```

   Everything will be installed into a new subdirectory named `apache-ant- + version`. You can rename the directory after the extraction if you wish.

2. Set the environmental variable `JAVA_HOME` to the base directory of your Java JRE or SDK, like

   ```
   export JAVA_HOME; JAVA_HOME=/usr/java/j2sdk1.4.0
   ```

   The location is entirely dependent upon your variety of UNIX. Sun's rpm distributions of Java normally install to `/usr/java/something`. Sun's System V package distributions of Java (including those that come with Solaris) normally install to `/usr/something`, with a sym-link from `/usr/java` to the default version (so for Solaris you will usually set JAVA_HOME to `/usr/java`).

3. Remove the existing file `HSQLDB_HOME /lib/hsqldb.jar`.

4. cd to `HSQLDB_HOME/build`. Make sure that the bin directory under your Ant home is in your search path. Run the following command.

   ```
   ant hsqldb
   ```

   This will build a new `HSQLDB_HOME/lib/hsqldb.jar`.

See the Building HSQLDB Jars appendix if you want to build anything other than `hsqldb.jar` with all default settings.

Setting up a HyperSQL Persistent Database Catalog and a HyperSQL Network Listener

If you installed from an OS-specific package, you may already have a catalog and listener pre-configured. See if your package includes a file named `server.properties` (make use of your packaging utilities). If you do, then I suggest that you still read this section while you poke around, in order to understand your setup.

1. Select a UNIX user to run the database process (JVM) as. If this database is for the use of multiple users, or is a production system (or to emulate a production system), you should dedicate a UNIX user for this purpose. In my examples, I use the user name `hsqldb`. In this chapter, I refer to this user as the `HSQLDB_OWNER`, since that user will own the database catalog files and the JVM processes.

   If the account doesn't exist, then create it. On all system-5 UNIXes and most hybrids (including Linux), you can run (as root) something like

   ```
   useradd -c 'HSQLDB Database Owner' -s /bin/bash -m hsqldb
   ```

   (BSD-variant users can use a similar `pw useradd hsqldb...` command).

2. Become the `HSQLDB_OWNER`. Copy the sample file `sample/server.properties` to the `HSQLDB_OWNER`'s home directory and rename it to `server.properties`. (As a final reminder, "sampleserver.properties" is a relative path, so it is understood to be relative to your `HSQLDB_HOME`).

   ```
   # Hsqldb Server cfg file.
   # See the HyperSQL Network Listeners chapter of the HyperSQL User Guide.

   # Each server.database.X setting defines a database "catalog".
   # I.e., an independent set of data.
   # Each server.database.X setting corresponds exactly to the jdbc:hsqldb:*
   ```

```
# JDBC URL you would use if you wanted to get a direct (In-Process)
# Connection to the catalog instead of "serving" it.

server.database.0=file:db0/db0
# I suggest that, for every file: catalog you define, you add the
# connection property "ifexists=true" after the database instance
# is created (which happens simply by starting the Server one time).
# Just append ";ifexists=true" to the file: URL, like so:
# server.database.0=file:db0/db0;ifexists=true

# server.dbname.0 defaults to "" (i.e. server.dbname.n for n==0), but
# the catalog definition n will be entirely ignored for n > 0 if you do not
# set server.dbname.n.  I.e. dbname setting is required for n > 0, though it
# may be set to blank (e.g. "server.dbname.3=")
```

Since the value of the first database (server.database.0) begins with *file:*, the catalog will be persisted to a set of files in the specified directory with names beginning with the specified name. Set the path to whatever you want (relative paths will be relative to the directory containing the properties file). You can read about how to specify other catalogs of various types, and how to make settings for the listen port and many other things in other chapters of this guide.

3. Set and export the environmental variable `CLASSPATH` to the value of `HSQLDB_HOME` (as described above) plus "/lib/hsqldb.jar", like

```
export CLASSPATH; CLASSPATH=/path/to/hsqldb/lib/hsqldb.jar
```

In `HSQLDB_OWNER`'s home directory, run

```
nohup java org.hsqldb.server.Server &
```

This will start the Listener process in the background, and will create your new database catalog "db0". Continue on when you see the message containing `HSQLDB server... is online`. `nohup` just makes sure that the command will not quit when you exit the current shell (omit it if that's what you want to do).

Accessing your Database

We're going to use SqlTool to access the database, so you will need the file `sqltool.jar` in addition to `hsqldb.jar`. If `sqltool.jar` isn't already sitting there beside `hsqldb.jar` (they both come pre-built), build it exactly as you would build `hsqldb.jar`, except use ant target `sqltool`. If your distribution came with a sqltool jar file with a version label, like `sqltool-1.2.3.4.jar`, that's fine-- use that file whenever I say `sqltool.jar` below.

Copy the file sample/sqltool.rc to the `HSQLDB_OWNER`'s home directory. Use `chmod` to make the file readable and writable only to `HSQLDB_OWNER`.

```
# $Id: sqltool.rc 6381 2021-11-18 21:45:56Z unsaved $

# This is a sample RC configuration file used by SqlTool, DatabaseManager,
# and any other program that uses the org.hsqldb.lib.RCData class.
# See the documentation for SqlTool for various ways to use this file.
# This is not a Java Properties file. It uses a custom format with stanzas,
# similar to .netrc files.

# If you have the least concerns about security, then secure access to
# your RC file.

# You can run SqlTool right now by copying this file to your home directory
# and running
#    java -jar /path/to/sqltool.jar mem
# This will access the first urlid definition below in order to use a
# personal Memory-Only database.
# "url" values may, of course, contain JDBC connection properties, delimited
# with semicolons.
# As of revision 3347 of SqlFile, you can also connect to datasources defined
# here from within an SqlTool session/file with the command "\j urlid".

# You can use Java system property values in this file like this:  ${user.home}

# Windows users are advised to use forward slashes instead of back-slashes,
# and to avoid paths containing spaces or other funny characters.  (This
# recommendation applies to any Java app, not just SqlTool).

# It is a runtime error to do a urlid lookup using RCData class and to not
# match any stanza (via urlid pattern) in this file.

# Three features added recently.  All are downward-compatible.
# 1. urlid field values in this file are now comma-separated (with optional
```

```
#    whitespace before or after the commas) regular expressions.
# 2. Each individual urlid token value (per previous bullet) is a now a regular
#    expression pattern that urlid lookups are compared to.  N.b. patterns must
#    match the entire lookup string, not just match "within" it.  E.g. pattern
#    of . would match lookup candidate "A" but not "AB".  .+ will always match.
# 3. Though it is still an error to define the same exact urlid value more
#    than once in this file, it is allowed (and useful) to have a url lookup
#    match more than one urlid pattern and stanza.  Assignments are applied
#    sequentially, so you should generally add default settings with more
#    liberal patterns, and override settings later in the file with more
#    specific (or exact) patterns.

# Since service discovery works great in all JREs for many years now, I
# have removed all 'driver' specifications here.  JRE discover will
# automatically resolve the driver class based on the JDBC URL format.
# Most people use default ports, so I have removed port specification from
# examples except for Microsoft's Sql Server driver where you can't depend
# on a default port.
# In all cases, to specify a non-default port, insert colon and port number
# after the hostname or ip address in the JDBC URL, like
# jdbc:hsqldb:hsql://localhost:9977 or
# jdbc:sqlserver://hostname.admc.com:1433;databaseName=dbname

# Amazon Aurora instances are access from JDBC exactly the same as the
# non-Aurora RDS counterpart.

# For using any database engine other than HyperSQL, you must add the
# JDBC jar file and the SqlTool jar to your CLASSPATH then run a command like:
#     java org.hsqldb.util.SqlTool...
# I.e., the "-jar" switch doesn't support modified classpath.
# (See SqlTool manual for how to do same thing using Java modules.)
# To oversimplify for non-developers, the two most common methods to set
# CLASSPATH for an executable tool like SqlTool are to either use the java
# "-cp" switch or set environmental variable CLASSPATH.
# Windows users can use graphical UI or CLI "set".  Unix shell users must
# "export" in addition to assigning.
#
# All JDBC jar files used in these examples are available from Maven
# repositories.  You can also get them from vendor web sites or with product
# bundles (especially database distributions).
# Most databases provide multiple variants.  Most people will want a type 4
# driver supporting your connection mechanism (most commonly TCP/IP service,
# but also database file access and others) and your client JRE version.
# By convention the variants are distinguished in segments of the jar file
# name before the final ".jar" .


# Global default.  .+ matches all lookups:
urlid .+
username SA
password

# A personal Memory-Only (non-persistent) database.
# Inherits username and password from default setting above.
urlid mem
url jdbc:hsqldb:mem:memdbid

# A personal, local, persistent database.
# Inherits username and password from default setting above.
urlid personal
url jdbc:hsqldb:file:${user.home}/db/personal;shutdown=true;ifexist=true
transiso TRANSACTION_READ_COMMITTED
# When connecting directly to a file database like this, you should
# use the shutdown connection property like this to shut down the DB
# properly when you exit the JVM.

# This is for a hsqldb Server running with default settings on your local
# computer (and for which you have not changed the password for "SA").
# Inherits username and password from default setting above.
# Default port 9001
urlid localhost-sa
url jdbc:hsqldb:hsql://localhost



# Template for a urlid for an Oracle database.
# Driver jar files from this century have format like "ojbc*.jar".
# Default port 1521
urlid localhost-sa
# Avoid older drivers because they have quirks.
# You could use the thick driver instead of the thin, but I know of no reason
# why any Java app should.
```

```
#urlid cardiff2
# Can identify target database with either SID or global service name.
#url jdbc:oracle:thin:@//centos.admc.com/tstsid.admc
#username blaine
#password asecret


# Template for a TLS-encrypted HSQLDB Server.
# Remember that the hostname in hsqls (and https) JDBC URLs must match the
# CN of the server certificate (the port and instance alias that follows
# are not part of the certificate at all).
# You only need to set "truststore" if the server cert is not approved by
# your system default truststore (which a commercial certificate probably
# would be).
# Port defaults to 554.

#urlid tls
#url jdbc:hsqldb:hsqls://db.admc.com:9001/lm2
#username BLAINE
#password asecret
#truststore ${user.home}/ca/db/db-trust.store


# Template for a Postgresql database
# Driver jar files are of format like "postgresql-*.jar"
# Port defaults to 5432.
#urlid blainedb
#url jdbc:postgresql://idun.africawork.org/blainedb
#username blaine
#password asecret

# Amazon RedShift (a fork of Postgresql)
# Driver jar files are of format like "redshift-jdbc*.jar"
# Port defaults to 5439.
#urlid redhshift
#url jdbc:redshift://clustername.hex.us-east-1.redshift.amazonaws.com/dev
#username awsuser
#password asecret

# Template for a MySQL database.  MySQL has poor JDBC support.
# The latest driver jar files are of format like "mysql-jdbc*.jar", but not
# long ago they were like "mysql-connector-java*.jar".
# Port defaults to 3306
#urlid mysql-testdb
#url jdbc:mysql://hostname/dbname
#username root
#password asecret
# Alternatively, you can access MySQL using jdbc:mariadb URLs and driver.

# Note that "databases" in SQL Server and Sybase are traditionally used for
# the same purpose as "schemas" with more SQL-compliant databases.

# Template for a Microsoft SQL Server database using Microsoft's Driver
# Seems that some versions default to port 1433 and others to 1434.
# MSDN implies instances are port-specific, so can specify port or instname.
#urlid msprojsvr
# Driver jar files are of format like "mssql-jdbc-*.jar".
# Don't use older MS JDBC drivers (like SQL Server 2000 vintage) because they
# are pitifully incompetent, handling transactions incorrectly.
# I recommend that you do not use Microsoft's nonstandard format that
# includes backslashes.
#url jdbc:sqlserver://hostname;instanceName=instname;databaseName=dbname
# with port:
#url jdbc:sqlserver://hostname:1433;instanceName=instname;databaseName=dbname
#username myuser
#password asecret

# Template for Microsoft SQL Server database using the JTDS Driver
# Looks like this project is no longer maintained, so you may be better off
# using the Microsoft driver above.
# http://jtds.sourceforge.net  Jar file has name like "jtds-1.3.1.jar".
# Port defaults to 1433.
# MSDN implies instances are port-specific, so can specify port or instname.
#urlid nlyte
#username myuser
#password asecret
#url jdbc:jtds:sqlserver://myhost/nlyte;instance=MSSQLSERVER
# Where database is 'nlyte' and instance is 'MSSQLSERVER'.
# N.b. this is diff. from MS tools and JDBC driver where (depending on which
# document you read), instance or database X are specified like  HOSTNAME\X.
```

```
# Template for a Sybase database
#urlid sybase
#url jdbc:sybase:Tds:hostname:4100/dbname
#username blaine
#password asecret
# This is for the jConnect driver (requires jconn3.jar).

# Derby / Java DB.
# Please see the Derby JDBC docs, because they have changed the organization
# of their driver jar files in recent years.  Combining that with the different
# database types supported and jar file classpath chaining, and it's not
# feasible to document it adequately here.
# I'll just give one example using network service, which works with 10.15.2.0.
# Put files derbytools*.jar, derbyclient*.jar, derbyshared*.jar into a
# directory and include the path to the derbytools.jar in your classpath.
# Port defaults to 1527.
#url jdbc:derby://server:<port>/databaseName
#username ${user.name}
#password any_noauthbydefault
# If you get the right classes into classpath, local file URLs are like:
#url jdbc:derby:path/to/derby/directory
# You can use \= to commit, since the Derby team decided (why???)
# not to implement the SQL standard statement "commit"!!
# Note that SqlTool can not shut down an embedded Derby database properly,
# since that requires an additional SQL connection just for that purpose.
# However, I've never lost data by shutting it down improperly.
# Other than not supporting this quirk of Derby, SqlTool is miles ahead of
# Derby's ij.

# Maria DB
# With current versions, the MySQL driver does not work to access a Maria
# database (though the inverse works).
# Driver jar files are of format like "mariadb-java-client*.jar"
# Port defaults to 3306
#urlid maria
#url jdbc:mariadb://hostname/db2
#username blaine
#password asecret
```

We will be using the "localhost-sa" sample urlid definition from the config file. The JDBC URL for this urlid is `jdbc:hsqldb:hsql://localhost`. That is the URL for the default catalog of a HyperSQL Listener running on the default port of the local host. You can read about URLs to connect to other catalogs with and without listeners in other chapters of this guide.

Run `SqlTool`.

```
java -jar path/to/sqltool.jar localhost-sa
```

If you get a prompt, then all is well. If security is of any concern to you at all, then you should change the privileged password in the database. Use the command `SET PASSWORD` command to change SA's password.

```
SET PASSWORD 'newpassword';
```

Set a *strong* password!

> **Note**
> If, like most UNIX System Administrators, you often need to make up strong passwords, I highly suggest the great little program `pwgen`. You can probably get it where you get your other OS packages. The command `pwgen -1` is usually all you need.

Note that with SQL-conformant databases like HyperSQL 2.0, user names and passwords are case sensitive. If you don't quote the name, it will be interpreted as upper-case, like any named SQL object. (Only for backwards compatibility, we do make an exception for the special user name SA, but you should always use upper-case "SA" nevertheless).

When you're finished playing, exit with the command `\q`.

If you changed the SA password, then you need to update the password in the `sqltool.rc` file accordingly.

You can, of course, also access the database with any JDBC client program. You will need to modify your classpath to include `hsqldb.jar` as well as your client class(es). You can also use the other HSQLDB client programs, such as `org.hsqldb.util.DatabasManagerSwing`, a graphical client with a similar purpose to `SqlTool`.

You can use any normal UNIX account to run the JDBC clients, including `SqlTool`, as long as the account has read access to the `sqltool.jar` file and to an `sqltool.rc` file. See the Utilities Guide about where to put `sqltool.rc`, how to execute sql files, and other `SqlTool` features.

Create additional Accounts

Connect to the database as SA (or any other Administrative user) and run `CREATE USER` to create new accounts for your catalog. HSQLDB accounts are database-catalog-specific, not `Listener`-specific.

In SQL-compliant databases, all database objects are created in a *schema*. If you don't specify a schema, then the new object will be created in the default schema. To create a database object, your account (the account that you connected with) must have the role `DBA`, or your account must have authorization for the target schema (see the CREATE SCHEMA command about this last). When you first create a HyperSQL catalog, it has only one database user-- SA, a DBA account, with an empty string password. You should set a password (as described above). You can create as many additional users as you wish. To make a user a DBA, you can use the "ADMIN" option to the `CREATE USER` command, command, or GRANT the DBA Role to the account after creating it.

Once an object is created, the object creator and users with the DBA role will have all privileges to work with that object. Other users will have only the rights which the pseudo-user PUBLIC has. To give specific users more permissions, even rights to read objects, you can GRANT permissions for specific objects, grant Roles (which encompass a set of permissions), or grant the DBA Role itself.

Since only people with a database account may do anything at all with the database, it is often useful to permit other database users to view the data in your tables. To optimize performance, reduce contention, and minimize administration, it is often best to grant SELECT to PUBLIC on table-like objects that need to be accessed by multiple database users, with the significant exception of any data which you want to keep secret. (Similarly with EXECUTE priv for routines and USAGE priv for other object types). Note that this is not at all equivalent to giving the world or the Internet read access to your tables-- you are giving read access to people that have been given accounts for the target database catalog.

Shutdown

Do a clean database shutdown when you are finished with the database catalog. You need to connect up as SA or some other Admin user, of course. With SqlTool, you can run

```
java -jar path/to/sqltool.jar --sql 'shutdown;' localhost-sa
```

You don't have to worry about stopping the `Listener` because it shuts down automatically when all served database catalogs are shut down.

Running Hsqldb as a System Daemon

You can, of course, run HSQLDB through inittab on System V UNIXes, but usually an init script is more convenient and manageable. This section explains how to set up and use our UNIX init script. Our init script is only for use by root. (That is not to say that the *Listener* will run as root-- it usually should not).

The main purpose of the init script is to start up a Listener for the database catalogs specified in your `server.properties` file; and to gracefully shut down these same catalogs. For each catalog defined by a `server.database.X` setting in your .properties file, you must define an administrative "urlid" in your `sqltool.rc` (these are used to access the catalogs for validation and shutdown purposes). Finally, you list the urlid names in your init script config file. If, due to firewall issues, you want to run a WebServer instead of a Server, then make sure you have a healthy WebServer with a webserver.properties set up, adjust your URLs in `sqltool.rc`, and set TARGET_CLASS in the config file.

By following the commented examples in the config file, you can start up any number of Server and/or WebServer listener instances with or without TLS encryption, and each listener instance can serve any number of HyperSQL catalogs (independent data sets), all with optimal efficiency from a single JVM process. There are instructions in the init script itself about how to run multiple, independently-configured JVM processes. Most UNIX installations, however, will run a single JVM with a single Listener instance which serves multiple catalogs, for easier management and more efficient resource usage.

After you have the init script set up, root can use it anytime to start or stop HSQLDB. (I.e., not just at system bootup or shutdown).

Portability of `hsqldb` init script

The primary design criterion of the init script is portability. It does not print pretty color startup/shutdown messages as is common in late-model Linuxes and HPUX; and it does not keep subsystem state files or use the startup/shutdown functions supplied by many UNIXes, because these features are all non-portable.

Offsetting these limitations, this one script does its intended job great on the UNIX varieties I have tested, and can easily be modified to accommodate other UNIXes. While you don't have tight integration with OS-specific daemon administration guis, etc., you do have a well-tested and well-behaved script that gives good, utilitarian feedback.

Init script Setup Procedure

The strategy taken here is to get the init script to run your single Server or WebServer first (as specified by TARGET_CLASS). After that's working, you can customize the JVM that is run by running additional Listener instances in it, running your own application in it (embedding), or even overriding HSQLDB behavior with your own overriding classes.

1. Copy the init script `sample/hsqldb.init` to `hsqldb` in the directory where init scripts live on your variety of UNIX. The most common locations are `/etc/init.d` or `/etc/rc.d/init.d` on System V style UNIXes, `/usr/local/etc/rc.d` on BSD style UNIXes, and `/Library/StartupItems/hsqldb` on OS X (you'll need to create the directory for the last).

2. View your `server.properties` file. Make a note of every catalog define by a `server.database.X` setting. A couple steps down, you will need to set up administrative access for each of these catalogs. If you are using our sample `server.properties` file, you will just need to set up access for the catalog specified with `file:db0/dbo`.

   > **Note**
   > Pre-2.0 versions of the hsqldb init script required use of .properties settings of the form`server.urlid.X`. These settings are obsolete and should be removed.

3. Either copy `HSQLDB_OWNER`'s `sqltool.rc` file into root's home directory, or set the value of `AUTH_FILE` to the absolute path of `HSQLDB_OWNER`'s `sqltool.rc` file. This file is read directly by root, even if you run hsqldb as non-root (by setting `HSQLDB_OWNER` in the config file). If you copy the file, make sure to use `chmod` to restrict permissions on the new copy. The init script will abort with an appropriate exhortation if you have the permissions set incorrectly.

   You need to set up a urlid stanza in your `sqltool.rc` file for network access (i.e. JDBC URL with hsql:, hsqls:, http:, or https:) for each catalog in your `server.properties` file. For our example, you need to define a stanza for the `file:db0/db0` catalog. You must supply for this catalog, a hsql: JDBC URL, an administrative user name, and the password.

   Example 16.1. example sqltool.rc stanza

   ```
   urlid localhostdb1
   url jdbc:hsqldb:hsql://localhost
   username SA
   password secret
   ```

4. Look at the comment towards the top of the init script which lists recommended locations for the configuration file for various UNIX platforms. Copy the sample config file `sample/hsqldb.conf` to one of the listed locations (your choice). Edit the config file according to the instructions in it. For our example, you will set the value of `URLIDS` to `localhostdb1`, since that is the urlid name that we used in the `sqltool.rc` file.

   ```
   # $Id: hsqldb.conf 6310 2021-02-28 15:25:00Z unsaved $

   # Sample configuration file for HyperSQL Server Listener.
   # See the "HyperSQL on UNIX" chapter of the HyperSQL User Guide.

   # N.b.!!!!  You must place this in the right location for your type of UNIX.
   # See the init script "hsqldb" to see where this must be placed and
   # what it should be renamed to.

   # This file is "sourced" by a Bourne shell, so use Bourne shell syntax.

   # This file WILL NOT WORK until you set (at least) the non-commented
   # variables to the appropriate values for your system.
   # Life will be easier if you avoid all filepaths with spaces or any other
   # funny characters.  Don't ask for support if you ignore this advice.

   # The URLIDS setting below is new and REQUIRED.  This setting replaces the
   # server.urlid.X settings which used to be needed in your Server's
   # properties file.

   # -- Blaine (blaine dot simpson at admc dot com)

   JAVA_EXECUTABLE=/usr/bin/java

   # Unless you copied the jar files from another system, this typically
   # resides at $HSQLDB_HOME/lib/sqltool.jar, where $HSQLDB_HOME is your HSQLDB
   # software base directory.
   # The file name may actually have a version label in it, like
   # sqltool-1.2.3.jar (in which case, you must specify the full name here).
   # A 'hsqldb.jar' file (with or without version label) must reside in the same
   # directory as the specified sqltool.jar file.
   SQLTOOL_JAR_PATH=/opt/hsqldb-2.0.0/hsqldb/lib/sqltool.jar
   ```

```
# For the sample value above, there must also exist a file
# /opt/hsqldb-2.0.0/hsqldb/lib/hsqldb*.jar.

# Where the file "server.properties" or "webserver.properties" resides.
SERVER_HOME=/opt/hsqldb-2.0.0/hsqldb/data

# What UNIX user the server will run as.
# (The shutdown client is always run as root or the invoker of the init script).
# Runs as root by default, but you should take the time to set database file
# ownerships to another user and set that user name here.
HSQLDB_OWNER=hsqldb

# The HSQLDB jar file specified in HSQLDB_JAR_PATH above will automatically
# be in the class path.  This arg specifies additional classpath elements.
# To embed your own application, add your jar file(s) or class base
# directories here, and add your main class to the INVOC_ADDL_ARGS setting
# below.  Another common use-case for adding to your class path is to make
# classes available to the DB engines for SQL/JRT functions and procedures.
#SERVER_ADDL_CLASSPATH=/usr/local/dist/currencybank.jar

# For startup or shutdown failures, you can save a lot of debugging time by
# temporarily adjusting down MAX_START_SECS and MAX_TERMINATE_SECS to a
# little over what it should take for successful startup and shutdown on
# your system.

# We require all Server/WebServer instances to be accessible within
# $MAX_START_SECS from when the Server/WebServer is started.
# Defaults to 60.
# Raise this is you are running lots of DB instances or have a slow server.
#MAX_START_SECS=200

# Max time to allow for JVM to die after all HSQLDB instances stopped.
# Defaults to 60.  Set high because the script will always continue as soon as
# the process has stopped.  The importance of this setting is, how long until
# a non-stopping-JVM-problem will be detected.
#MAX_TERMINATE_SECS=0

# NEW AND IMPORTANT!!!
# As noted at the top of this file, this setting replaces the old property
# settings server.urlid.X.
# Simply list the URLIDs for all DB instances which your *Server starts.
# Usually, these will exactly mirror the server.database.X settings in your
# server.properties or webserver.properties file.
# Each urlid listed here must be defined to a NETWORK url with Admin privileges
# in the AUTH_FILE specified below.  (Network type because we use this for
# inter-process communication)
# Separate multiple values with white space.  NO OTHER SPECIAL CHARACTERS!
# Make sure to quote the entire value if it contains white space separator(s).
URLIDS='localhostdb1'

# These are urlids # ** IN ADDITION TO URLIDS **, for instances which the init
# script should stop but not start.
# Most users will not need this setting.  If you need it, you'll know it.
# Defaults to none (i.e., only URLIDS will be stopped).
#SHUTDOWN_URLIDS='ondemand'

# SqlTool authentication file used only for shutdown.
# The default value will be sqltool.rc in root's home directory, since it is
# root who runs the init script.
# (See the SqlTool chapter of the HyperSQL Utilities Guide if you don't
# understand this).
#AUTH_FILE=/home/blaine/sqltool.rc

# Typical users will leave this unset and it will default to
# org.hsqldb.server.Server.  If you need to run the HSQLDB WebServer class
# instead, due to a firewall or routing impediment, set this to
# org.hsqldb.server.WebServer, see the docs about running WebServr, and
# set up a "webserver.properties" file instead of a "server.properties".
# The JVM that is started can invoke many classes (see the following item
# about that), but this is the server that is used (1) to check status,
# (2) to shut down the JVM.
#TARGET_CLASS=org.hsqldb.server.WebServer

# This is where you may specify both command-line parameters to TARGET_CLASS,
# plus any number of additional progams to run (along with their command-line
# parameters).  The MainInvoker program is used to embed these multiple
# static main invocations into a single JVM, so see the API spec for
# org.hsqldb.util.MainInvoker if you want to learn more.
# N.b. You should only use this setting to set HSQLDB Server or WebServer
# parameters if you run multiple instances of this class, since you can use the
# server/webserver.properties file for a single instance.
# Every additional class (in addition to the TARGET_CLASS)
```

```
# must be preceded with an empty string, so that MainInvoker will know
# you are giving a class name.  MainInvoker will invoke the normal
# static main(String[]) method of each such class.
# By default, MainInvoker will just run TARGET_CLASS with no args.
# Example that runs just the TARGET_CLASS with the specified arguments:
#INVOC_ADDL_ARGS='-silent false'   #but use server.properties property instead!
# Example that runs the TARGET_CLASS plus a WebServer:
#INVOC_ADDL_ARGS='"" org.hsqldb.server.WebServer'
# Note the empty string preceding the class name.
# Example that starts TARGET_CLASS with an argument + a WebServer +
# your own application with its args (i.e., the HSQLDB Servers are
# "embedded" in your application).  (Set SERVER_ADDL_CLASSPATH too).:
#INVOC_ADDL_ARGS='-silent false "" org.hsqldb.server.WebServer "" com.acme.Stone --env prod localhost'
#   but use server.properties for -silent option instead!
# Example to run a non-TLS server in same JVM with a TLS server.  In this
# case, TARGET_CLASS is Server which will run both in TLS mode by virtue of
# setting the tls, keyStore, and keyStorePassword settings in
# server*.properties, as described below; plus an "additional" Server with
# overridden 'tls' and 'port' settings:
#INVOC_ADDL_ARGS="'' org.hsqldb.server.Server --port 9002 --tls false"
# This is an important use case.  If you run more than one Server instance,
# you can specify different parameters for each here, even though only one
# server.properties file is supported.
# Note that you use nested quotes to group arguments and to specify the
# empty-string delimiter.

# The TLS_* settings have been obsoleted.
# To get your server running with TLS, set
# system.javax.net.ssl.keyStore=/path/to/your/private.keystore
# system.javax.net.ssl.keyStorePassword=secretPassword
# server.ssl=true
# IN server.properties or webserver.properties, and
# MAKE THE FILE OWNER-READ-ONLY!
# See the TLS Encryption section of the HyperSQL User Guide, paying attention
# to the security warning(s).
# If you are running with a private server cert, then you will also need to
# set "truststore" in the your SqlTool config file (location is set by the
# AUTH_FILE variable in this file, or it must be at the default location for
# HSQLDB_OWNER).

# Any JVM args for the invocation of the JDBC client used to verify DB
# instances and to shut them down (SqlToolSprayer).
# Server-side System Properties should normally be set with system.*
# settings in the server/webserver.properties file.
# This example specifies the location of a private trust store for TLS
# encryption.
# For multiple args, put quotes around entire value.
# If you are starting just a TLS_encrypted Listener, you need to uncomment
# this so the init scripts uses TLS to connect.
# If using a private keystore, you also need to set "truststore" settings in
# the sqltool.rc file.
#CLIENT_JVMARGS=-Djavax.net.debug=ssl
# This sample value displays useful debugging information about TLS/SSL.

# Any JVM args for the server.
# For multiple args, put quotes around entire value.
#SERVER_JVMARGS=-Xmx512m
# You can set the "javax.net.debug" property on the server side here, in the
# same exact way as shown for the client side above.
```

Verify that the init script works.

Just run

```
/path/to/hsqldb
```

as root to see the arguments you may use. Notice that you can run

```
/path/to/hsqldb status
```

at any time to see whether your HSQLDB `Listener` is running.

Re-run the script with each of the possible arguments to really test it good. If anything doesn't work right, then see the Troubleshooting the Init Script section.

5. Tell your OS to run the init script upon system startup and shutdown. If you are using a UNIX variant that has `/etc/rc.conf` or `/etc/rc.conf.local` (like BSD variants and Gentoo), you must set "hsqldb_enable" to "YES" in either of those files. (Just run `cd /etc; ls rc.conf rc.conf.local` to see if you have one of these files). For good UNIXes that

use System V style init, you must set up hard links or soft links either manually or with management tools (such as `chkconfig` or `insserv`) or GUIs (like run level editors).

This paragraph is for Mac OS X users only. If you followed the instructions above, your init script should reside at `/Library/StartupItems/hsqldb/hsqldb`. Now copy the file `StartupParameters.plist` from the directory `src/org.hsqldb/sample` of your HSQLDB distribution to the same directory as the init script. As long as these two files reside in `/Library/StartupItems/hsqldb`, your init script is active (for portability reasons, it doesn't check for a setting in `/etc/hostconfig`). You can run it as a *Startup Item* by running

```
SystemStarter {start|stop|restart} Hsqldb
```

Hsqldb is the service name. See the man page for `SystemStarter`. To disable the init script, wipe out the `/Library/StartupItems/hsqldb` directory. Hard to believe, but the Mac people tell me that during system shutdown the Startup Items don't run at all. Therefore, if you don't want your data corrupted, make sure to run "SystemStarter stop Hsqldb" before shutting down your Mac.

Follow the examples in the config file to add additional classes to the server JVM's classpath and to execute additional classes in your JVM. (See the `SERVER_ADDL_CLASSPATH` and `INVOC_ADDL_ARGS` items).

Troubleshooting the Init Script

Definitely look at the init script log file, which is at an OS-sependent location, but is usually at `/var/log/hsqldb.log`.

Do a `ps` to look for processes containing the string `hsqldb`, and try to connect to the database from any client. If the init script starts up your database successfully, but incorrectly reports that it has not, then your problem is with specification of urlid(s) or SqlTool setup. If your database really did not start, then skip to the next paragraph. Verify that your config file assigns a urlid for each catalog defined in `server.properties` or `webserver.properties`, then verify that you can run `SqlTool` as root to connect to the catalogs with these urlids. (For the latter test, use the `--rcfile` switch if you are setting `AUTH_FILE` in the init script config file).

If your database really is not starting, then verify that you can `su` to the database owner account and start the database. The command `su USERNAME -c ...` won't work on most UNIXes unless the target user has a real login shell. Therefore, if you try to tighten up security by disabling this user's login shell, you will break the init script. If these possibilities don't pan out, then debug the init script or seek help, as described below.

To debug the init script, run it in verbose mode to see exactly what is happening (and perhaps manually run the steps that are suspect). To run an init script (in fact, any sh shell script) in verbose mode, use `sh` with the `-x` or `-v` switch, like

```
sh -x path/to/hsqldb start
```

See the man page for `sh` if you don't know the difference between `-v` and `-x`.

If you want troubleshooting help, use the HSQLDB lists/forums. Make sure to include the revision number from your `hsqldb` init script (it's towards the top in the line that starts like "# $Id:"), and the output of a run of

```
sh -x path/to/hsqldb start > /tmp/hstart.log 2>&1
```

Upgrading

This section is for users who are using our UNIX init script, and who are upgrading their HyperSQL installation.

Most users will not have customized the init script itself, and your customizations will all be encapsulated in the init script configuration file. These users should just overwrite their init script with a new one from the HyperSQL installation, and manually merge config file settings. First, just copy the file `/sample/hsqldb.init` over top of of your init script (wherever it runs from). Then update your old config file according to the instructions in the new config file template at `sample/hsqldb.conf`. You will have to change very few settings. If you are upgrading from a pre-2.0 installation to a post-2.0 installation, you will need to (1) add the setting `URLIDS`, as described above and in the inline comments, and (2) replace variable `HSQLDB_JAR_PATH` with `SQLTOOL_JAR_PATH` which (if you haven't guessed) should be set to the path to your `sqltool.jar` file.

Users who customized their init script will need to merge their customizations into the new init script.

$Revision: 6752 $