Shore Storage Manager Installation Manual[1]

The Shore Project Group
Computer Sciences Department
UW-Madison
Madison, WI

Version 2.0

August 25, 1999

[1]This research is sponsored by the Advanced Research Project Agency, ARPA order number 018 (formerly 8230), monitored by the U.S. Army Research Laboratory under contract DAAB07-92-C-Q508. Further funding for this work was provided by DARPA through Rome Research Laboratory Contract No. F30602-97-2-0247.

Contents

1 Introduction

The Storage Manager Version 2.0 distribution consists of documentation and source files. This document describes how to install the documentation and source files and how to build the Storage Manager from the sources.

2 Requirements

To install the documentation and source files and build the software, you need:

- a Sparcstation or Intel X86 processor
- UNIX : Solaris (we use SunOS 5.6) or Linux (we use RedHat Linux 2.0.36 - For various Linux distributors, see http://www.redhat.com/, http://www.debian.org/, http://www.slackware.com/, and http://www.suse.com/); with at least 8 MB bytes of *shared memory*
  OR
  Windows NT 4.0 with Cygwin installed (we use Cygwin_nt-3.0 20.1 - see http://www.cygnus.com/) and with Visual C++ 6.0 (and all its service packs) installed.
- Perl 5 (we use 5.004_04) See http://www.activestate.com/ActivePerl/ and http://www.perl.com/.
- You will also need the following, which come with Cygwin on NT and with RedHat Linux:
  - gzip file (de)compression facility to unpack the release.
  - GNU make.
  - The Tcl library (version 8.0). (We have heard reports of problems using the version that comes wiht Cygwin. If you have trouble, install the Scriptics version.)
  - The normal Unix utilities like `csh, sh, mv, ar, ld, tar, sed`, etc.
- The following C++ compiler:
  - on Unix, gcc 2.7.2.3 or egcs (we use version 1.0.3)
  - on NT (regardless of the build environment you use), you must have Visual C++ 6.0 installed to compile and link the Storage Manager
- A PDF reader or printer for reading the documentation.
- Lots of free disk space. The disk space requirements are roughly:
  - documents (files in both PDF and PS forms): 4.9 MB
  - sources alone: 9 MB.
  - sources, docs, and space to build from scratch:

* Release version (without debugging symbols and without trace code): (75 MB on NT, 135 MB on X86, 165 MB on Sparcstation)
* Debug version (with debugging symbols and with trace code): (110 MB on NT, 100 MB on X86, 100 MB on Sparcstation)

The GNU software for Unix is available via ftp from ftp://prep.ai.mit.edu/pub/gnu or ftp://gatekeeper.dec.com/pub/GNU.

The GNU software for NT comes with the Cygwin environment, which is available from http://sourceware.cygnus.com/cygwin/

If you are unfamiliar with Tcl, it stands for "tool command language". It was developed by Dr. John Ousterhout at UC-Berkeley for providing interfaces for tools, although it can be used for more than that. It is used in a test shell for the Storage Manager. You can build the Storage Manager without the tester, in which case you do not need Tcl.

Tcl is available from http://www.scriptics.com/.

3 Warnings

We recommend that you *not* use virus-detection programs with the Storage Manager, as they intercept I/O requests and may cause problems that are difficult to diagnose; they will certainly deteriorate performance. For similar reasons, we recommend that logs and data volumes be placed on *local disks*.

4 How to Use These Installation Instructions

You can build this software in the following environments (and possibly others, but we have tested only these). The steps for building in these environments have the labels given below. When following the instructions in this manual, do not succumb to the temptation to skip over the prose between the labelled steps. The labels only serve to distinguish steps that differ among environments.

UNIX You can build on Linux or Solaris using the GNU utilities (with the egcs or gcc compiler).

NT-VS You can build on Windows NT using the Visual Studio development environment, with the Visual C++ compiler and linker. The project files in the release build both Debug and Release versions by default. In both versions, the Storage Manager libraries are static (not dynamic link libraries).

NT-CYG You can build on Windows NT using the GNU utilities provided with the Cygwin installation. This uses GNU Make, but compiles and links with the native C++ compiler and linker (distributed with Visual C++).[1]

[1] You are welcome to modify the configuration to use the GNU utilities, but we have not tested this and do not provide instructions for doing so. You will have to decipher our configuration scheme, which is admittedly arcane. You should start by looking at the file `config/shore.def` in the source release.

To install this software using these instructions, you must use a Unix-like shell. (You can install the software on NT without such a shell, but these instructions do not help you with that.) For NT, we suggest that you use the Bash shell window that comes with Cygwin (when installed, its icon is labelled "Cygwin B20"). In our experience, some implementations of `tcsh` for NT do not work with our make files.

The installation instructions refer to these two shell variables:

```
set TARDIR=directory-that-holds-tar-files
set SMROOT=directory-in-which-to-install-sm
```

If you use the Bourne shell (`/bin/sh`) or one of the shells derived from it, such as `ksh` or `bash`, omit the word "set":

```
TARDIR=directory-that-holds-tar-files
SMROOT=directory-in-which-to-install-sm
```

The directory `$TARDIR` should already contain the tar files `doc-2.0.tar.gz` and `src-2.0.tar.gz` (or `prj-2.0.tar.gz` if you plan to build on Windows NT using the Microsoft Visual Studio environment).

Whatever shell you use, make a directory in which to build and install the Storage Manager.

UNIX **NT-VS** **NT-CYG**

```
mkdir -p $SMROOT
```

5 Installing the Documentation Release

You must have copied the tar files with FTP into the directory `$TARDIR`.

UNIX **NT-VS** **NT-CYG**

```
mkdir -p $SMROOT/doc
cd $SMROOT/doc
gunzip -c $TARDIR/doc-2.0.tar.gz | tar xvf -
```

This installs a `doc` directory containing the following subdirectories and files. Both PostScript and PDF versions are included.

`installation.{pdf,ps}` This document.

`README` Contains copyright information.

`smdoc.{pdf,ps}` Hacker's guide to the source code.

`ssmapi.{pdf,ps}` Brief description of the programmer's interface (API) to the Storage Manager.

`manfc/` Manual pages (in Unix man style) for the code found in `src/fc`, foundation classes (this has nothing to do with Microsoft foundation classes).

`mancommon/` Manual pages (in Unix man style) for the code found in `src/common`, more code likely to be common to client and server sides.

`mansthread/` Manual pages (in Unix man style) for the code found in `src/sthread`, the Shore threads package.

`manssm/` Manual pages (in Unix man style) for the code found in `src/sm`, the Storage Manager proper.

`thread_debug` An ASCII text file describing how to switch threads when debugging with the GNU debugger, gdb.

6 Building The Storage Manager

This section describes how to compile and link the Storage Manager. The first thing you must do is to determine the paths to the utilities you you will use so that you can set up your environment variables. If you do not know where things are installed on your machine, use the *Find* tool to identify the paths needed by searching for the items given below.

`cl.exe` The native compiler.

`bash.exe` The Cygwin tools.

`perl.exe` Perl.

`STDLIB.H` The system include files.

`MSVCRT.LIB` The C libraries.

6.1 Preparing your Environment

After adjusting the three environment variables (Path, INCLUDE, LIB, and MAKE_MODE) by following the instructions below, you must open a new shell to continue your work with the new path.

6.1.1 PATH environment variable

To build the system, `perl, make, tar,` and the normal Unix utilities must be in your PATH environment variable. If you are building on NT (whether **NT-CYG** or **NT-VS**), be sure the Cygwin utilities (e.g., `bash.exe`) are also in your path.

UNIX

```
# for Bourne and its family of shells :
PATH=$PATH:<locations of Perl, GNU utils>

# for C and its family of shells :
setenv PATH=$PATH:<locations of Perl, GNU utils>
```

NT-VS

Click on *Start - Settings - Control Panel - System - Environment*. In the list of *User Variables*, click on *Path* and in the *Value* window, add the location of Perl and the Cygwin tools. You must open a new shell to continue your work with the new path.

NT-CYG

If you are building with GNU make under Cygwin, your path environment variable must contain the path to the native compiler (Visual C++) and linker executables, `cl.exe, link.exe`, as well as the paths to Perl and the Cygwin tools.

Click on *Start - Settings - Control Panel - System - Environment*. In the list of *User Variables*, click on *Path* and in the *Value* window, add the location of Perl, the Cygwin utilities and the VC++ compiler.

6.1.2 INCLUDE environment variable

When you build on NT (regardless of the build environment) using the native VC++ compiler the compiler needs to #include system include files. It locates these by the paths in the `INCLUDE` environment variable.

NT-VS **NT-CYG**

Click on *Start - Settings - Control Panel - System - Environment*. In the list of *User Variables*, click on *INCLUDE* and in the *Value* window, add the paths to the compiler's include files (e.g., `STDLIB.H`).

6.1.3 LIB environment variable

When you build on NT, some of the dynamic link libraries have to be located. The `LIB` environment variable takes care of that.

NT-VS **NT-CYG**

Click on *Start - Settings - Control Panel - System - Environment.* In the list of *User Variables*, click on *LIB* and in the *Value* window, add the paths to the C run-time libraries (e.g., `MSVCRT.LIB`).

6.1.4 MAKE_MODE environment variable

When you build on NT with Cygwin Make, the `MAKE_MODE` environment variable should be set to "UNIX".

NT-VS **NT-CYG**

Click on *Start - Settings - Control Panel - System - Environment.* In the panel labelled *Variable*, enter `MAKE_MODE`, and in the panel labelled *Value*, enter `UNIX`.

6.2 Installing the Sources

First, un-`tar` the release somewhere. Go to a directory where you want to install the sources. For simpilicity, we let that be `$SMROOT`. Users of `sh, ksh`, etc. should omit the word "set":

UNIX **NT-CYG**

```
set SRCDIR=$SMROOT/build
mkdir -p $SRCDIR
cd $SRCDIR
gunzip -c $TARDIR/src-2.0.tar.gz | tar xvf -
```

If you build with NT-VS, instead install the project and source files:

NT-VS

```
set SRCDIR=$SMROOT/build
```

```
mkdir -p $SRCDIR
cd $SRCDIR
gunzip -c $TARDIR/prj-2.0.tar.gz | tar xvf -
```

This installs the sources as described below, as well as a directory `$SRCDIR/nt_buildsm`, which contains the Visual Studio project files.

Now you have the following files and directories in `$SRCDIR`:

`GNUmakefile` A bootstrapping Makefile, which invokes `imake`.

`Imakefile` Source from which Makefile is generaged by `imake`.

`imake/` Sources for `imake` and Perl script to bootstrap building of `imake`.

`config/` Configuration files and other files read by `imake`.

`tools/` Perl scripts (`*.pl`) used by the build environment, including those that generate source files (`*_gen.{cpp,h}`) in `src/`.

`src/` Sources for Storage Manager, including inputs files (`*.dat`) to Perl scripts for generating source files.

After you have followed the directions below to build the Storage Manager, you will also find `$SRCDIR/installed`.

6.3 Setting Build-time Configuration Options

Before you build the Storage Manager, you must give values to some configuration options. The options are located in the file `$SRCDIR/config/shore.def`.

> **NOTE** The file `shore.def` is #included by the Makefiles, so it controls what files are built and what tools are used when building under `make`, but not when building under Visual Studio. It is *also* #included by the sources, so it controls conditional compilation in the sources, regardless what build tools are used to build the software.

UNIX **NT-VS** **NT-CYG**

The release contains several sample `shore.def` files for different configurations. Choose the one that best suits your environment.

```
cd $SRCDIR/config
ls shore.def.*

#  choose the ARCH and VERS to suit your environment.
set ARCH=linux
set VERS=debug

#  Use the appropriate shore.def
cp shore.def.$ARCH.$VERS  shore.def

# You MUST edit shore.def to provide the correct values for the
# PERL_DIR_PATH (see below).

# You MUST edit shore.def to provide correct values for the
# TCL_DIR_PATH, TCL_DIR_VERSION and TCL_LIB_EXTENSION (see below)
# or else you must undefine BUILD_DEFAULT_SSH (see below)
```

UNIX **NT-VS** **NT-CYG**

The file `shore.def` contains numerous C-preprocessor macros that affect configuration and compilation. In the list of macros below, only the ones marked with an asterisk (*) are required for building under Visual Studio.

Normally the only ones you care about are those described below. The options controlling debugging support and optimization take one of the values `ON` and `OFF`. We describe them briefly here. More information about the effects of these options can be found by reading the comments in `shore.def`. Some of the "options" are not optional (they do not work). For exampoe, `USE_COORD` assumes the existence of code that is not distributed in this release.

`DEBUGCODE` controls whether auditing and assert checking code is generated. For development and testing purposes we strongly suggest setting this to `ON`. Turning it on seems to slow things down by a factor of two, at least.

`TRACECODE` controls whether tracing code is installed. For development and testing purposes we strongly suggest setting this to `ON`. Turning it on seems to slow things down by a factor of two, at least.

`DEBUGGERSYMBOLS` controls the generation of symbols for use by debuggers (`-g` flag for `gcc`).

`OPTIMIZE` controls compiler flags related to optimization. When `OPTIMIZE` is `OFF` and you are compiling with `gcc`, `gcc` is called with `-O` because this seems to avoid `gcc` bugs (at least in earlier versions of `gcc`).

`TCL_DIR_PATH` You must define this to point to the location of your installed TCL if you choose to build the tester, `ssh`. To build the name of the library, the build system appends

/lib/lib*VER***ext** to this path, where *VER* is the value of TCL_DIR_VERSION, below, and **ext** is the value of TCL_DIR_EXTENSION, below.

TCL_DIR_VERSION If you choose to build the tester (ssh), you must define this to give the version number in the name of the Tcl libraries in your installation. If your library is installed as libtcl8.0.so, you should define TCL_DIR_VERSION to be 8.0.

TCL_DIR_EXTENSION If you choose to build the tester (ssh) on Unix, you must define this to determine which Tcl library will be linked with ssh. On NT, this is ignored. On Unix, this must be .a for the static library or it must be .so for the dynamic load library. If it is undefined, the static library is used. Note the ".".

INSTALL_PATH If you want make install to install anywhere other than $SMROOT/installed/, set this. This has no effect on builds with Visual Studio.

*PERL_DIR_PATH Location of Perl (needed to build Makefiles and generated sources). The build system appends /bin/perl to this path. *This MUST be set to the proper path before you can create the generated source files (a prerequisite to building, regardless which environment you are using.*

The following options are simply #define-d or #undef-ed.

BUILD_DEFAULT_SSH If defined, the Storage Manager tester shell ssh is built automatically (this requires Tcl). You can undefine it to avoid building ssh[2].

BUILD_DEFAULT_SMLAYER_TESTS You can define this to cause the default make target to descend into the various tests/ directories and build the unit tests. WARNING: this consumes lots of disk space!

FORCE_EGCS Undefine this if you use gcc Version 2.7.3 rather than egcs.

*DONT_TRUST_PAGE_LSN If this is defined, formatting of volumes is faster at the expense of longer recovery, as page formats must be redone during recovery. If this is undefined, the LSNs on pages are trusted to have been cleared when the volume was formatted, which means that when volumes are formatted, each page on the volume is initialized. With large volumes, this takes a long time.

NT-VS

For building with Visual Studio, some of the configuration options are in the project files, and since there are many projects that make up the Storage Manager, it is not easy to change these options. For this reason, we have put as many configuratin options as possible in source files that are included during compilation and generated by the build system.

[2]This shell has nothing to do with the other ssh, the Secure Shell.

The path to Tcl *must* be provided if the tester `ssh` is to be built, but it must be provided by editing files other than `config/shore.def`.

Now you must edit two files to locate your installed Tcl. (These files are *not* generated or edited by the build process. It is best if you make a backup copy of these files before you edit them.)

Both files are in `$SRCDIR/nt_buildsm/src/smlayer/sm/ssh`:

`ssh.dsp` Locate two instances of the string `tcl80.lib` and edit the paths as appropriate to your installation of Tcl.

`local_flags.i` Change the last line to reflect the path for your installation of Tcl.

This editing can be done with the `sed` script `fix_tcl.sh`, which can be *sourced* in a `bash` shell, or the files can be hand-edited. If you use the script, you must first edit the variables `TCLINCLUDE` and `TCLLIB` appropriately. Be careful to escape backslashes in the paths.

6.4 Creating Generated Files

When building with `make`, you must generate the Makefiles and several source files[3] with `imake`. The source directory contains `imake`, along with a Perl script to bootstrap `imake`.

This step must be taken even if you are building with Visual Studio, because the tar file containing the source and the project files does not contain the generated files. See the section Omitting `ssh` *below.*

NT-VS **NT-CYG**

```
# Do not do these steps on Unix.
cd $SRCDIR
make for_vstudio
```

Under Unix, this "make" step is optional, and the target is `automatic` rather than `for_vstudio`; it is optional because it is accomplished by the compilation step (in the next subsection).

6.5 Building the Storage Manager

Now that you have installed the sources (and project files, if building with Visual Studio) and generated any necessary files, you can build the libraries. The distribution is configured to build the libraries and the Storage Manager tester shell `ssh`. If you do not wish to build `ssh`, see the section Omitting `ssh` below.

[3] Generated source files are called `*_gen.h` and `*_gen.cpp`

NT-VS

Using the Explorer, navigate to the directory `$SRCDIR/nt_buildsm/src` and *click on* `src.dsw` to open the workspace with Visual C++. The first time you open the workspace, you will see a dialog box indicating that Visual Studio cannot find one or more of the components of the workspace. This refers to the `.ncb` file, which is generated by Visual Studio if it is missing. [4]

Having opened the workspace, *click the following sequence: Build - Batch Build - Build.* This builds both the Debug and Release versions of the Storage Manager and its tester, and it takes 2-3 hours. If you wish to build only one or the other, see Building only the Debug or Release Version, below.

UNIX **NT-CYG**

```
cd $SRCDIR
make
```

This takes 30 - 60 minutes, depending on the architecture and your choice of versions to build (Debug or Release).

6.6 Selecting Subsets to Build

6.6.1 Omitting `ssh`

If you do not wish to build the tester (`ssh`):

UNIX **NT-CYG**

Undefine `BUILD_DEFAULT_SSH` in the configuration file `shore.def` and (re-)start with the step Creating Generated Files, above.

NT-VS

Having opened the workspace with Visual C++, *click the following sequence: Build - Batch Build.* Locate the two project conigurations for ssh in the list (`ssh - Win32 Debug` and `ssh - Win32 Release`) and click on them. The check marks next to them should disappear. Now *click on Build.*

---

[4] You must have installed Visual C++/Visual Studio 6.0!

6.6.2 Building only the Debug or Release Version

If you wish to build only the Debug or Release versions of the Storage Manager:

UNIX **NT-CYG**

In the Unix and Cygwin environments, only one of the versions (Debug or Release) is built. If it is not the one you want, do one of the following:

- Go back to the step Setting Configuration Options and choose a different `shore.def` file. Continue from there.
- Change the values for `DEBUGCODE`, `TRACECODE`, and `OPTIMIZE` in the file `$SRCDIR/config/shore.def`, as described in the section Setting Configuration Options, above. Continue with the step Creating Generated Files, above.

NT-VS

Having opened the workspace with Visual C++, *click the following sequence: Build - Batch Build.* Locate the all project conigurations labelled *xxx-* `Win32 Debug` and click on them. The check marks next to them should disappear. Now *click on Build.*

6.6.3 Regenerating Source Files

If you choose to modify code and wish to re-build from scratch, regardless of your build environment, you can do so. Even if you are building with Visual Studio, you *must* use the Cygwin utilties to regenerated the (generated) source files.

If you want to change the configuration or if you are modifying sources, you can use

`make make` to create the Makefiles.

`make automatic` on UNIX, to generate Makefiles and generated source files.

`make for_vstudio` on NT, to generate Makefiles and generated source files.

`make depend` to make dependencies. (UNIX, NT-CYG)

`make clean` to remove all object and executable files. (UNIX, NT-CYG)

`make pristine` to remove all generated files. (UNIX, NT-CYG)

6.6.4 Building Unit Tests

The Storage Manager comes with unit tests for each library. The directories containing the source for each library also contains a subdirectory called `tests/`. *By default, these tests are not built.* You can build them as follows:

UNIX **NT-CYG**

Descend to the `tests` directory of choice. Run

```
make automatic
make
```

This builds all the tests in that directory. You can build a test `foo` (`foo.exe` on NT) with

```
make foo  # make foo.exe on NT
```

NT-VS

Open Visual Studio on the workspace `ntbuild_sm/src/src.dsw`. Choose the project whose name matches the test of interest. Select it with a right click, make it the active project if necessary, and *click Build (selection only)* . Remember that there are two versions of each project, a Debug and a Release version. The easiest way to choose one or the other is by using the sequence *Build - Set Active Configuration*, select the item in the list of *Project configurations*, then use any of the myriad ways to build the selected project.

If you wish to build one or more test programs for the threads layer, descend to `src/smlayer/sthread/tests`. One test of interest is `ioperf`: It can be used to measure I/O performance through the threads package. Below is its usage information.

```
(prompt): ioperf
Usage: ioperf [-s block_size] [-n block_count] [-R random] [-l local_io] [-f
fastpath_io] [-k keepopen_io] [-c check_flag] [-r read_only] [ -w write_only]
[-b read_and_write] file
```

`-s block_size` Determines the size of a write or read request, as in `dd`.

`-n block_count` Determines the total number of blocks read or written.

`-R` If used, `ioperf` issues seeks to pseudo-random locations within the range [0,block_count].

`-l` If used, `ioperf` performs I/O requests locally rather than through the `diskrw` process [Unix only].

`-f` Ignore. (Not implemented.)

-k Ignore.

-c Perform internal sanity check on seek cursor.

-r Read the file.

-w Write the file.

-b Read and write the file.

file Name of file (or device).

Before you can run the program under Unix, it must have access to the `diskrw` program. (On NT, the diskrw functions run in a thread of the program rather than in a separate process.)

For example (UNIX) :

```
# locate diskrw
ln -s ../diskrw

# create a file called "junk"
ioperf -s 8192 -n 1000 -w junk

# read junk (sequentially)
ioperf -s 8192 -n 1000 -r junk

# read junk (pseudo-randomly)
ioperf -s 8192 -n 1000 -R junk
```

7 Installing The Storage Manager

UNIX **NT-CYG**

The command `make install` installs the executable files in `$SRCDIR/installed/bin`, the include files in `$SRCDIR/installed/include`, and other supporting files in `$SRCDIR/installed/lib`.

```
cd $SRCDIR
make install
```

To install elsewhere, simply move this directory, or change the target location by updating the macro `INSTALL_PATH` in `shore.def` before you build. See the section Setting Configuration Options, above.

NT-VS

In a Bash shell, you can source the script install_all. If you want to change the destination directory, edit this script. You must also edit the script to choose between Debug and Release versions.

```
cd $SRCDIR
#
# edit install_all to change the default target of the install
#
source install_all
```

8 Testing Your Installation

This section lists steps to test your installation. Included are painfully brief instructions for running the Storage Manager tester. We do not include complete documentation for this tester; rather, we just give a single command to run a large set of test scripts.

8.1 Running ssh

The Storage Manager shell (tester) is built with the include files and libraries in the source tree (not with the installed Storage Manager). The directories in which ssh is built is the easiest place to run ssh.

UNIX **NT-CYG**

```
cd $SRCDIR/src/smlayer/sm/ssh
```

NT-VS

```
cd $SRCDIR/nt_buildsm/src/smlayer/sm/ssh
```

Now create directories in which to store the data and log, and copy (and edit, if you like) the configuration file for the Storage Manager and the startup file for ssh. NOTE: The buffer pool must be fairly large (3.2 MB) to get through all the tests, because some of the tests “hog” buffer pool frames.

UNIX **NT-CYG** **NT-VS**

```
mkdir -p volumes log
cp .shoreconfig.example .shoreconfig
cp .sshrc.example .sshrc
```

You are ready to run `ssh`. Be sure that in the context in which you run `ssh`, `./volumes` and `./log` are on a *local disk*. Ssh can be run on a short script just to verify that things are properly built, or it can be run on a full set of scripts (this takes several hours with a Debug version, and an hour or more with a Release version).

UNIX **NT-CYG**

```
# short test:
./ssh -f scripts/vol.init
devid_t::devid_t(./volumes/dev1): open:
1. error in sdisk_unix.cpp:157 Operating system error  --- No such file or directory
# You can safely ignore the above messages (if you get it), which is
# a warning message issued when no log files are encountered at
# recovery time.
# Under NT, the message is slightly different: "The system cannot find the
# file specified."

# long test:
./ssh -f scripts/all
```

NT-VS

```
# short test:
Release/ssh.exe -f scripts/vol.init
# The first time you run this, you should see a message
# akin to the one shown above (in the Unix case).

# long test:
Release/ssh.exe -f scripts/all
```

8.2 Running `file_scan`

In `src/smlayer/sm/tests` is a sample program that populates and scans a file. The directory contains a sample makefile, `Makefile.gcc` for use with `gcc`. This make file builds with the

installed Storage Manager include files and libraries. You must already have taken the steps in the section Installing The Storage Manager, above.

Edit the Makefile.gcc to refer to the correct `INSTALL_DIR`. *It is also important to set the* `GCC` *macro to use the same compiler used to build the Storage Manager.*

The file `exampleconfig` is the run-time configuration file read by `file_scan`. Be sure that in the context in which you run `file_scan`, `./volumes` and `./log` are on a *local disk*.

UNIX **NT-CYG**

```
cd $SRCDIR/src/smlayer/sm/tests

# Edit Makefile.gcc as needed

make -f Makefile.gcc
mkdir -p log volumes

file_scan -i
# Ignore message about no such file or directory.

file_scan
```

You can build this test with Visual Studio, but the installed workspace does not build against the libraries and include files that you just installed. It builds with those found in the source directories. If you want to build against those you installed, you must change the dependencies and files associated with the project.

NT-VS

```
cd $SRCDIR/nt_buildsm/src/smlayer/sm/file_scan
mkdir -p log volumes
```

Open the workspace with Visual C++. *Click the following sequence: Build - Batch Build.* Locate the two project conigurations for file_scan in the list (`file_scan - Win32 Debug` and `file_scan - Win32 Release`) and click on one or the other. The check mark next to it should appear. Now *click on Build.* After it is built, *click the following sequence: Build - ! Execute file_scan.exe.*

---

9 Setting Run-time Configuration Options

The Storage Manager requires the setting of configuration options, which is normally done by reading a configuration file in the the `main` program. The sources for `ssh` show how this is done. The sample configuration file `$SRCDIR/src/smlayer/sm/ssh/.shoreconfig.example` shows the format of configuration options.

A complete list of the run-time configuration options is in the sources, in `src/smlayer/sm/sm.cpp`. In that file, each call to the function `add_option` creates a run-time option. There is also a manual page describing run-time options; the manual page is `manssm/ssm_options.pdf`.