CTN Software Installation Guide

Guide to Installing Software on UNIX and Windows Computers

Stephen M. Moore
David E. Beecher
Nilesh R. Gohel

Mallinckrodt Institute of Radiology
Electronic Radiology Laboratory
510 South Kingshighway Boulevard
St. Louis, Missouri 63110
314/362-6965 (Voice)
314/362-6971 (FAX)

Version 2.10.3
February 8, 1999

This document provides installation instructions and test procedures for CTN software. These instructions will help you extract the software from a CD ROM or a file from an ftp site and run some simple tests to verify the software is operating properly.

1 Introduction

This manual describes procedures for installing the CTN demonstration software developed by the Electronic Radiology Laboratory (ERL) at the Mallinckrodt Institute of Radiology (MIR). This software was developed on a SUN SPARCStation 10 under SunOS 4.1.3 and under Solaris 2.x. It has been compiled and tested under other machines that will be listed below. This manual discusses the installation procedure for each of these machines and identifies any machine-dependent issues. This manual also provides a description of the software libraries, applications and other source files present with the system and procedures used to test the installation.

2 Machine Specific Details

The CTN software is written in ANSI C and has been compiled on several different machines: Sun SPARCStation 10, SunOS 4.1.3, Solaris 2.3, Solaris 2.4, Solaris 2.5.1 DEC Alpha workstation, OSF/1 3.0 Some users have contributed changes for compiling on AIX, HPUX and Linux systems. The previous release was compiled on an IRIX system. Since we do not have access to these three environments, you may find some problems compiling on these machines.

We are in the process of modifying the code to run in the Win32 environment (NT 4.0, Windows 95). This is the first (beta) release of that software.

This section identifies any machine-dependent issues that we have encountered.

2.1 Sun SPARCStation 10, Sun OS 4.1.3

The standard C compiler supplied by Sun was not ANSI compliant. Sun does supply an ANSI C compiler (acc) which is ANSI compliant. We used acc to compile the CTN software.

Sun supplies “include” files with acc that are supposed to be ANSI compliant (that is, supply proper function prototypes). We believe some of the files supplied by Sun are in error (they don’t match our references) and we changed the following files on our SPARCstations:

/usr/lang/SC2.0.1/include/cc_411/stdlib.h
    changed definition of malloc to:

```
extern void *malloc(size_t);
```

/usr/lang/SC2.0.1/include/cc_411/string.h
    modified several function definitions to:

```
extern void *memchr(const void*, int, size_t);
extern int memcmp(const void*, const void*, size_t);
extern void *memcpy(void*, const void *, size_t);
extern void *memmove(void *, const void *, size_t);
extern void *memset(void *, int, size_t);
```

The version of the kernel delivered with the SPARCstation does not have the shared memory and semaphore support required by the GQ routines. You may need to reconfigure your system if that is not already installed. Use the Unix `ipcs` utility to check for shared memory and semaphore support.

We used a third party supplier of Motif for our SunOS environment. We also used a third party supplier of X11 because we needed X11R5 for our Motif applications. The version of X-Windows that was delivered with SunOS was X11R4. If you are using SunOS, you should check with Sun about X11R5 and Motif. If Sun cannot supply this, there are other vendors who can.

We have also compiled this software successfully using `gcc` and SunOS 4.1.3.

2.2 Sun SPARCStation 10, Solaris 2.3

The standard C compiler supplied by Sun was not ANSI compliant. Sun provides the same ANSI C compiler under Solaris 2.1 as it does for Sun OS 4.1.3. Under Solaris, the ANSI C compiler supersedes the K&R C compiler. Therefore, you install the ANSI C compiler, modify your path accordingly and invoke the compiler as `cc`. We did not have to modify the include files under Solaris to compile our software.

Solaris does come with X11R5. We had to install Motif on our Solaris machines and used the product that is sold by Sun. We do our software development in a Berkeley environment. Thus, our path normally includes `/usr/ucb` near the front to catch the Berkeley version of programs (like install and cc). We have modified the environment scripts to use explicit path names for programs like this that cause problems, but we may not have caught all of them. If you are working on a Solaris machine and run into strange behavior when compiling or installing with our software, check your path and scripts.

We use shared memory and semaphores for some types of process communication. This required some patches under Solaris 2.3 The patch ID to acquire and install is as follows:

Patch-ID# 101520-02
Keywords: semaphore optimizations ipcs Synopsis: SunOS 5.3: ipcs/semaphore fixes Date: Aug/02/94
Solaris Release: 2.3
SunOS release: 5.3
Topic: SunOS 5.3: ipcs/semaphore fixes
BugId's fixed with this patch: 1145619 1137598 1122375 1149376 1146296 1137598 1168646
Changes incorporated in this version: 1168646
Relevant Architectures: sparc
Patches accumulated and obsoleted by this patch:
Patches which conflict with this patch:
Patches required with this patch:
Obsoleted by:
Files included with this patch: /usr/bin/ipcs /kernel/sys/semsys
Problem Description: 1168646 ipcs does not reliably return information about all semaphores

ipcs(1) does not reliably return information about all allocated semaphore segments. The semaphore ID assignment algorithm was changed in 5.3 patch 101520, but ipcs(1) wasn't changed to match, so when a semaphore ID is assigned out of the previously usual sequence (greater than 65535), ipcs wouldn't show it.

(from 101520-01)
1145619 SVR4 semaphore operations could be optimized 1137598 semaphore code-optimizations (sem.c) 1122375 Sys V semaphore locks has too much contention 1149376 Memory leak in SVR4 semaphores 1146296 PANIC in System V IPC/Semaphore code 1137598 semaphore code-optimizations (sem.c)

2.3 Sun SPARCStation 20, Solaris 2.4

When we upgraded to Solaris 2.4, we installed the Sparcworks compiler and had no compile problems.

2.4 DEC Alpha workstation, OSF/1

We believe the standard C compiler supplied by DEC is ANSI compliant; we made no changes to the compilation environment. Our version of OSF/1 (V2.0) came with X11R5 and Motif. We made no changes to compile and run our applications.

2.5 Silicon Graphics workstation, IRIX

This information is based on experience with SGI machines running 5.x releases of Irix. We are likely well behind the current release of the OS.

We had a report from someone using Sybase on an SGI about installation problems. The messages said:

```
ninit: t_open, No  such device or address
ninit: All master network listeners have failed.
```

Sybase says this is a known problem with SGI and that you need to install an SGI patch to correct the problem. The SGI patch is EOE1.sw.svr4net

We don't have more explicit information about what version of IRIX this applies to or how to obtain the patches from Silicon Graphics.

There are also some libraries that Silicon Graphics uses for network operations that appear in libnsl.a. If you find unresolved references to `t_errno` or `t_open`, you will want to include the switch `-lnsl` in the macro LIBS_OS for the SGI environment. We also had a report about a problem with the SGI implementation of gethostbyname. To get around this, the user included the switches `-lc -lnsl` on the link line (in that order).

Since these reports, we have compiled a version of the software under Irix 5.2 and using mSQL. We have not run extensive tests, but we were able to get a clean compile.

2.6 Linux

We have compiled the software on Linux and run some simple tests. We ran our tests using the Caldera Red Hat version. We have received documentation and scripts from two users which should make the installation easier for people who are doing this for the first time or are not Linux experts. We have included this information in the contributed directory. We are not going to redocument it here.

2.7 Win32

We are compiling the software using version 4.2 of the Microsoft Visual C++ compiler and version 6.5 of the Microsoft SQL server. We have not seen any issues with either the compiler or database.

3 Database Issues

Some of the CTN demonstration applications require the use of relational database. In the original implementation, this was accomplished through the use of Sybase. In this version of the CTN software, we have included an implementation that can use Sybase or miniSQL on Unix systems and Microsoft SQL Server 6.5 on Windows NT systems. Users can choose the appropriate version through the use of the proper environment files.

The remainder of the section contains notes about the installation of miniSQL and Sybase. This does not serve as a substitute for the documentation for those products. We do not include precompiled libraries or executables for either database product.

3.1 MiniSQL Installation Notes

The author of miniSQL has changed his licensing terms. We feel that we can no longer distribute the miniSQL code with our software. Your organization will need to obtain the software and abide by the licensing rules. The ftp site is bond.edu.au. miniSQL is now at version 2 (or higher). We have email from a user who said he only needed to change the scripters for creatating tables. We have included the changes in `cfg_scripts/msql2` but have not tested it yet. We are running version 1.0.16.

There are a number of steps that must be performed in order to successfully install MiniSQL (msql) on your particular machine. These steps are enumerated below:

1. You should definitely print out the file `.../doc/mSQL-1.0.ps`. This is a copy of the user/admin manual and is very useful.
2. In general, follow the instructions in the README file in the MiniSQL install directory for compiling, installing, and testing msql.
3. You should create an msql user, and that user should be a member of the group that owns the CTN files (as released, group ID 100). Log in as msql to perform the following tasks. Database administration can only be performed as the msql user (please remember this fact).

4. Obtain the msql software from the ftp site listed above. Please pay the licensing fee.
5. Follow directions in README file for the make, setup, and install of msql. We suggest you use the default installation directory, `/usr/local/Minerva`. It also tells you how to start the server, etc. Be advised that when you start the server (msqld&) you will probably get a message that says:
   ```
   Couldn't open ACL file: No such file or directory
   ```
   This is ok. It just means that all the msql databases created will be globally available to all users. If you want to restrict usage, there are sample acl files around which will tell you how to do that, but for the purposes of testing, it probably just easier to leave access open.
6. Proceed to the directory `...../msql-version/src/tests` in the mSQL distribution.
7. Before the above can be run, you will need to compile and link `insert_test` and `select_test` in the directory msql-version/src/msql. For some reason, they weren't automatically built in the make process, but it is quite easy manually with a straight cc (or acc or gcc). Don't forget to specify the msql library in the compile line (libmsql.c). So...something like:
   ```
   [ag]cc -o insert_test insert_test.c <libdir>libmsql.a
   [ag]cc -o select_test select_test.c <libdir>libmsql.a
   ```
8. The "killer" script should now run.
9. The "rtest" script should also run.
10. If all this goes well, this should confirm that msql is alive and well.
11. You will have to make small changes to the script files `killer` and `rtest` to correctly specify locations for executables, etc. Please remember that as a rule, *we have not changed any of this release*, and we have had very few problems with the system to this point. Be advised that building and testing the system may be slightly different than you are used to with the DICOM distribution developed at MIR.

3.2 Sybase Installation Notes

There are a number of steps that must be performed in order to successfully install Sybase on your particular machine. The majority of these steps are outlined in the *Sybase Installation Guide* and will not be repeated in this section. If you are not using Sybase, the scripts that are supplied to you will be unusable except to examine them for functionality before porting them to your particular database environment.

After successfully installing Sybase, there are a couple of items that need some special attention. First, you must add a new user to the Sybase system called `sybase` with a password of `sybase`. All libraries supplied access the databases(s) with this user name and password. You must also be sure that you have successfully created devices on your system that Sybase can use for database storage. These device descriptions will of course vary depending on your system configuration. The script `CreateDevice.template`, located in `../cfg_scripts/Sybase/MakeDevices`, will be of considerable use. You may need to be logged in under the `sybase` user account to use these scripts. The applications can be run from other accounts because they have code to use the sybase account.

Note: It may be OK to use the account name `sybase`, but it is a bad idea to use `sybase` as the password. for that account. We strongly suggest that you use a different password. for the account.
Sybase itself has a password for each user that is different than the system password. By default, our software uses the name `sybase` and the password `sybase`

to access the SQL server. If you want to use a different password for sybase, you need to change the constant SYBASE_PASSWORD which is defined in `facilities/tbl/tbl_sybase.h`

A problem that you will most likely experience using Sybase is one of running out of user connections at some point. This is relatively easy to repair using the following commands. Get into the interactive SQL interpretor (`isql -Usa`), and execute the command `sp_configure`.

The two parameters to look for are “user connections” and “memory”. Both of these will most likely need to be increased and this is accomplished as follows:

```
sp_configure “memory”, 8192
go
sp_configure “user connections”,50 (or 100)
go
reconfigure
go
shutdown (you need to restart the server)
go
```

This procedure will set up the new parameters and restart the server so they can take effect. Your memory is most likely defaulted to 4096 and therefore doubling it is a reasonble thing to do. The user connections parameter is probably set at 20 or so and is actually the resource you are running out of. Be careful, you can’t (shouldn’t) increase the user connections without increasing the memory, or your server may not restart....so be sure you get the memory increased.

3.3 MS SQL Server Installation Notes

The MS SQL Server product provides a setup tool that guides you through the installation process. When the installation is complete, you will be able to interact with the server using interactive SQL and their GUI-based management tools. The CTN software communicates with the server using ODBC, and your PC must be configured to use that channel. Check the ODBC configuration to make certain it is correct.

Open the Control Panel folder and then open ODBC. Select the “System DSN” tab. There will be a list of system data sources. There should be one called LocalServer which uses the SQL Server driver. This entry is created by the SQL Server installation procedure. Our server is configured with these values (set by the SQL Server setup program).

Data Source Name	LocalServer
Description:	
Server:	(local)
Network Address:	(Default)
Network Address:	(Default)
[ ]	Use Trusted Connection

The CTN runs on the same machine as the SQL Server, so we use the local connection. We do not use the trusted connection option, but you might decided to do so depending on how you want to configure your system.

4 X11/Motif Installation Notes

All of the Motif-based applications in this release were designed with the commercially available interface builder, UIM/X. These applications automatically look for the file XKeysymDB in the directory /usr/lib/X11. If it isn't found, you will receive (many) warning messages of the following type when an application is started:

```
.
.
Warning: translation table syntax error: Unknown keysym name: osfAc-
tivate
Warning: ...found while parsing '<Key>osfActivate:ArmAndActivate()'
.
```

While these messages are not fatal to the application, they are a bother, and can be alleviated by placing the proper keysym definition file in `/usr/lib/X11/XKeysymDB.` This keysym file is typically a part of standard X11/Motif distribution, but may simply be placed in a different location than the one specified previously. The color definition file `/usr/lib/X11/rgb.txt` should also be present.

5 Installation Procedure

The installation procedure consists of several steps:

- Extract files from distribution CD ROM or ftp site.
- Select database package by using proper environment file (or by modifying an existing file).
- Compile some or all of the library modules.
- Move libraries to destination directory.
- (Optionally) compile some or all of the applications.
- Move binaries to destination directory.
- Run test programs.

5.1 Extract Files from Distribution

The software is distributed in both tar and zip formats. You will find these files on the distribution CD from the RSNA or on an ftp site (ftp.erl.wustl.edu, ftp.rsna.org).

Once the software has been extracted, you should find these directories in DICOM_HOME:

apps	Contains source code for all applications delivered with this system. You will find a number of subdirectories which contain individual applications.
bin	Contains target directories for binary versions of our applications. You will find separate directories under bin for each machine that we support, but you will find no binaries. You may choose to install your binaries in these directories or in a different directory (like `/usr/local/bin`).
contributed	Contains scripts, instructions, software contributed by users.
cfg_scripts	Contains configuration scripts and data used to setup the databases needed for CTN applications.
environments	Contains short scripts which set up the environments needed to compile and link the software. You will modify these files to fit your system.
facilities	Contains subdirectories with the sources for each of the major subroutine libraries.
include	Contains the common include files defined for each facility.
lib	Contains the target directories for different machine architectures. The software release does not include precompiled libraries.

5.1.1 Unix: Extract Files from Distribution

The files on the CD have these user and group IDs:

User	300	dicom
User	301	msql
Group	100	dicom

These are the user (uid) and group (gid) used at ERL. The specific numbers we chose are unimportant to you. You should decide who should own these files (owner) and which group should be used. We chose a dummy account (dicom) to own the files. Developers and users of the software are granted access by being members of the proper group. To extract the files:

- Choose owner and group for files and establish appropriate uid and gid.
- Create a home directory for the distribution. We will refer to this as DICOM_HOME. Set owner/group of DICOM_HOME to uid/gid.
- Become owner by `su owne`r or login as owner.
- Set current directory to DICOM_HOME:
  ```
  cd DICOM_HOME
  ```
- Mount the CD ROM on your machine. Assuming the mount point is /cdrom, extract files with tar: `tar xvf /cdrom/rsna97#1/ctn/ctn-2.9.0.tar.` (Note: later versions of the software may have a different version number.) Create msql account (if you want to install msql software)

These steps should extract the files from the distribution CD. They will be owned by owner (with user ID uid) and will have a group ID which is owner's group ID. If owner's group ID is `gid`, all of the files should now have the proper user IDs and group IDs. If the files are not extracted with the proper IDs (or you decide to use different IDs), you can change the IDs with the Unix `chown` command.

The software is delivered with read and write protections turned on for the owner and group members. "Other" users are granted read permission.

5.1.2 Windows: Extract Files from Distribution

Use your favorite Windows zip program to extract the files. The files do not have to be extracted to a specific directory, but we will assume you extract them to `C:\ctn` for the sake of this document.

5.2 Compilation Environment

The first step in compiling the libraries or binaries is establishing your environment. This allows you to :

- Identify target directories
- Define compiler options
- Define machine specific options

5.2.1 Unix: Compilation Environment

The `environments` directory contains several directories. Each directory is targeted at a specifc operating system (sunos, solaris, osf). In the OS-specific directories are pairs of files that define the compilation environment Examples are:

solaris.2.x.msql.gcc.noopt.env	make.solaris.2.x.msql.gcc.noopt
solaris.2.x.msql.noopt.env	make.solaris.2.x.msql.noopt

make.os.options is a file which is included by all Makefiles for the purpose of compiling the subroutine libraries (and applications). It defines a global set of rules to be used during the make process. The make environment has two sets of macros. One set of macros is used to define the existence of features in your system. If a macro is defined to the compiler, certain parts of the CTN code will be compiled and operational. A second set of macros defines other switches and constants that control the general system environment. Table 1 lists the macros that turn on features in the CTN code. Table 2 lists the other macros that control the compilation environment.

The best approach is to examine the existing files and to modify them to fit your environment.

TABLE 1: Macros Defined for Enabling CTN Features

Macro Name	**Definition**
DEBUG	Turns on some additional print/debug information in facilities. This code will be tripped when you place a facility in “debug” mode.
BIG_ENDIAN_ARCHITECTURE	Needs to be defined on big-endian on big-endian machines.
LITTLE_ENDIAN_ARCHITECTURE	Needs to be defined on big-endian on little-endian machines.
SHARED_MEMORY	Required for shared memory operation in queueing functions.
SEMAPHORE	Required for semaphore operations in queueing functions.
USLEEP	If the usleep function is provided by your operating system.
SYBASE	Defined if your system has sybase installed.
MSQL	Defined if your system has miniSQL
USEREGCOMP	Defined if you want to use the regcomp and regexec functions. These are found on HP and other systems.
SNOOP	Turns on SNP facility, DULsnoop extension, and DICOM snooper applications (Note: These will only work in a Solaris 2.x environment on Sun equipment.
CTN_NO_RUNT_PDVS	In some instances, the CTN can generate 0-length PDVs when writing data over the network. Turning on this option will perform some additional runtime tests to eliminate this. This was defined to satisfy one vendor who believes that 0-length PDVs are illegal. We don’t agree, but added the software as an option.
CTN_USE_THREADS	Define this compile time macro to make the CTN code thread safe. Note: This is currently under test with Solaris and Windows and is not guaranteed to produce thread safe code.

TABLE 2: Macros Defined for Controlling the Compilation Environment

Macro Name	**Definition**
LIBPATH_X11	Switch passed to linker for pathname for X11 libraries. Probably need not be defined unless your X11 libraries are not in a standard location.
LIBPATH_MOTIF	Switch passed to linker for pathname for Motif libraries. Probably need not be defined unless your Motif libraries are not in a standard location.
LIBPATH_UCB	Switch passed to linker for pathname for UCB libraries that are required for some operations (socket) under Solaris.
LIBPATH_DATABASE	Switch passed to linker for pathname for database libraries. This can be the pathname for sybase libraries or for a different database product.
LIBS_X11	Switches passed to linker to tell it which libaries to search for applications that use X11 (not the path to the libraries).

TABLE 2: Macros Defined for Controlling the Compilation Environment

Macro Name	**Definition**
LIBS_MOTIF	Switches passed to linker to tell it which libraries to search for applications that use Motif (not the path to the libraries).
LIBS_XAW	Switches passed to linker to tell it which libraries to search for applications that use the Athena widget set (not the path to the libraries).
LIBS_OS	Switches passed to linker to tell it to search any libraries that are dependent on the OS. We found this important for some libraries under Solaris 2.x.
LIBS_DATABASE	Switches passed to linker to tell it to search libraries to resolve database references. This is the switch you would use to tell the linker to search the libraries supplied by sybase to resolve the TBL references to sybase functions (or miniSQL).
LIBS_CTN	The list of libraries provided in the CTN software in order needed to resolve refererences. This is a convenience macro that makes it simple for someone to link an application (without having to remember each CTN library).
OS	An environment variable that we pass to makefiles to define the operating system. Will get used for conditional compilation. Values that are supported are: AIXV3 HPUX IRIX OSF SOLARIS SUNOS ULTRIX.
CFLAGS_X11	Any flags passed to compiler when compiling X11 applications. In some environments, this is a -I switch to give the location of include files.
CFLAGS_MOTIF	Any flags passed to compiler when compiling Motif applications. In some environments, this is a -I switch to give the location of include files.
LONGSIZE	The size of a variable (in bits) of type `long` on your system. Needs to be defined for our code to compile.
INTSIZE	The size of a variable (in bits) of type `int` on your system. Needs to be defined for our code to compile.
SHORTSIZE	The size of a variable (in bits) of type `short` on your system. Needs to be defined for our code to compile.
C_OPTS	The concatenation of a number of options to be passed to the compiler. Makefiles are expected to set CFLAGS equal to $(C_OPTS) plus whatever local options are needed.

`solaris.2.x.msql.gcc.noopt.env`, `solaris.2.x.msql.noopt.env`, and `solaris.2.x.sybase.noopt.env` are files which are used to establish the environment for building the software. These files contain “setenv” commands for the csh and should be used by users of the csh as follows:

```
      source solaris.2.x.msql.gcc.noopt.env
```

If you wish to use a different shell, you may need to alter the syntax in the file and how you use the file.

These environment files contain variables that define path names for files and target directories. It is safest to make these absolute path names.

DICOM_ROOT	The root directory for the installation. Most other directories are defined from this point.

DICOM_BIN	The location of the compiled binaries for the system. This will be the target directory when you rebuild the applications. Our destination directory is DICOM_HOME/bin/OS. You may choose to install your binaries somewhere else (e.g. `/usr/local/bin`).
DICOM_LIB	The location of the compiled library files. This will be the targetdirectory when you rebuild the libraries. Our destination directory is DICOM_HOME/lib/OS. You may choose to install your libraries somewhere else (e.g. `/usr/local/lib`). This is also used by our Makefiles when linking applications.
DICOM_INCLUDE	The location of the common include files for the subroutine libraries. Our Makefiles use this variable but also require that the include directory is DICOM_HOME/include. (The “make” supplied by one of our computer vendors has a problem with the VPATH variable.) Please use DICOM_HOME/include for this variable.
DICOM_MAKE	The path name to a file which is included by all Makefiles in the system. This allows us to set a number of global make options.

These environment files contain other variables which define how the code is to be compiled. Theses are:

ARCHITECTURE	One of LITTLE_ENDIAN_ARCHITECTURE or BIG_ENDIAN_ARCHITECTURE.
OS	One of AIXV3, IRIX, HPUX, OSF, ULTRIX, SUNOS or SOLARIS. This variable is used in make.os for machine dependent compilation.
CC	The C compiler to use to compile libraries and applications. This should be an ANSI compliant C compiler. If your system’s cc is ANSI compliant, you do not need to define this variable (its default is CC).

5.2.2 Windows: Compilation Environment

The compilation environment is managed within MSVC++. The one environment variable to set is DICOM_BIN. This is the destination directory for the binaries and is used by a make file to distribute the binaries are compiling/linking in the interactive environment.

The windows compilation environment uses the include file `dicom_platform.h` to define macros for controlling compile features. The file that is distributed has values set for Pentium machines, the Micro SQL Server and multi-threaded proesses. The macro definitions are shown in Table 3.

TABLE 3: Macros Defined for Compiing in a Windows Environment

Macro Name	**Value**	**Description**
LONGSIZE	32	The size of a variable of type long on your system. Needs to be defined for our code to compile.
INTSIZE	32	The size of a variable (in bits) of type int on your system. Needs to be defined for our code to compile.
SHORTSIZE	16	The size of a variable (in bits) of type short on your system. Needs to be defined for our code to compile.
LITTLE_ENDIAN_ARCHITECTURE		Defined because the Intel machines are little-endian
TBL_SQLSERVER		Defined to use Microsoft SQL server. If your site does not have this, remove this definition.
CTN_USE_THREADS		Defined to allow thread-safe applications. This needs to be defined if you will use some of the server applications, like `archive_server`.

5.3 Compiling and Installing Libraries

5.3.1 Unix: Compiling and Installing Libraries

To compile the libraries, you will need to specify a target directory where they are “installed”. We place our libraries in DICOM_HOME/lib/OS. For example, we use:

```
/dicom1a/projects/dicom94/lib/sunos
```

You may choose to install the libraries somewhere else (`/usr/local/lib`). You may need special privileges to install these files if you do not install them under DICOM_HOME.

The `facilities` directory contains these directories:

cfg	fis	icon	manage	snp
condition	gq	icpy	messages	sq
database	hap	idb	motif_utl	tbl
ddr	his	iman	objects	thread
dicom	hunks	info_entity	print	uid
dulprotocol	iap	lst	services	utility

The `facilities` directory contains a `Makefile` which can be used to make all of the libraries. In addition, each subdirectory contains its own `Makefile` for making that particular facility. Each `Makefile` has four “targets” of interest:

clean	Removes any object or library modules.
install	Builds the library (if needed) and installs it in $(DICOM_LIB).

indent	Runs the indent program to enforce coding style.
checkin/checkout	Checks code into or out of RCS.

To build and install the libraries, type

```
make install
```

from the `facilities` directory. This target is also used in the individual Makefiles, so you can reinstall one library by invoking make in the proper subdirectory.

5.3.2 Windows: Compiling and Installing Libraries

These instructions are based on our use of the MSVC++ environment. The library sources are collected in a single directory to ease the task of compiling. The compiling step will produce a single ctn library.

1. Start the Developer's Study and selet *Open Workspace*. Filter on `.mak` files and open:
   `C:\ctn\libwindows\ctnlib.mak`
2. Select the debug or release version. We have run both.
3. Select *project->settings*. Select the *C/C++* tab and pick the *Code Generation* pulldown. On that page, select *Multithreaded* or *Debug Multithreaded* in the *Use run-time library* box. Some of the server applications are multi-threaded.
4. Compile the library. You will get a number of warning messages, but should not get any fatal errors.

5.4 Compiling and Installing Applications

5.4.1 Unix: Compiling and Installing Applications

The procedure for building the applications is similar to building the libraries. You will need to set up the environment to identify the location of the library modules (used during the link phase) and the destination for binary files to be installed.

A `Makefile` is included in the apps directory which is used to make all of the applications. Each subdirectory has a `Makefile` for making the individual application(s) in a subdirectory. Each `Makefile` has the following "targets":

clean	Remove any object, core and binary files.
application	Compile and link the application.
install	Build the application (if necessary) and install it in $(DICOM_BIN).
indent	Passes code through indent program to enforce coding style.
checkin/checkout	Check code into or out of RCS.

To make all the applications, type

```
make applications
```

from the apps directory.

To make and install all of the applications, type

```
make install
```

from the apps directory.

These targets are also included with individual Makefiles in the subdirectories and can be used to build or install selected applications.

The TBL facility provides a standardized interface to at least one commerical database (sybase) and to a public domain database (miniSQL). When applications are linked, the user needs to specify which database is to be used. This is done in the master makefile found in the environments directory by defining macros as listed in Table 2. For example, macro definitions to enable sybase are:

```
LIBPATH_DATABASE = -L/usr/sybase/lib
LIBS_DATABASE = -ltbl_sybase -lsybdb
```

These macros instruct all application Makefiles where to find the sybase libraries (supplied by sybase) and the names of the libraries to resolve tbl and sybase function calls. LIBPATH_DATABASE defines where the linker will find libsybdb.a.

The macro definitions when using the miniSQL implementation are:

```
LIBPATH_DATABASE = -L/usr/local/Minerva/lib
LIBS_DATABASE = -ltbl_msql -lmsql
```

5.4.2 Windows: Compiling and Installing Applications

Applications are built using the MSVC++ environment. Open the workspace `C:\ctn\apps\win32apps.mak`. You will probably need to tell the system where to find include files for the CTN libraries and where to find the CTN library file. Select *tools->options* and examine the options for include files and library files. The include files should have `C:\ctn\include`. The library path should include `C:\ctn\libwindows\debug` (if you built the debug library). Once the parameters are set, you can build individual applications or build all of them by selecting *win32apps - Win32 Debug* or *win32apps - Win32 Release*.

You may need to change the compile options for applications if they do not link properly. As described in Section 5.3.2, select *Debug Multithreaded* or *Multithreaded* in the *Code Generation/ Use run-time library* drop down.

When the applications have been compiled and linked, the executables are left in various subdirectories. You can pull them together and install them in DICOM_BIN using a mak file in the apps directory:

```
nmake /f win_install.mak debug_install
```

This copies the debug version to the directory pointed at by the environment variable

DICOM_BIN. `nmake` is one of the applications that is bundled with MSVC++ and should be in your path.

5.5 Run Time Notes

5.5.1 Windows: Run Time Notes

Applications that use the SQL Server database make use of an environment variable to get access to the database. The environment variable SQL_ACCESS is used when tables are opened to determine the database server, login name and password. The format is:

SERVER:login:password:

We use the ODBC connection to LocalServer and use the sa account for access. Therefore, we set this variable to:

LocalServer:sa:<password>:

You might choose to use a different server naming convention or use different login names for SQL access. You create the login name and password using the SQL Server tools. *sa* is the System Administrator account that comes with the system. You can use this account or create another.

If you choose to use the Trusted Server feature of the system, you can leave the login and password values blank in the SQL_ACCESS variable: “LocalServer:::”.

The SQL Server has security features that allow the administrator to restrict access to tables in the database. You may find that you need to open up access to get the CTN software to operate properly. Run the CTN programs as described below or in the *User’s Guide*. If they complain about access privileges, you will need to use the *SQL Enterprise Manager* to give you access rights to the databases (insert, delete). There are several methods for allowing access. One simple method is to activate the Manage pulldown (in SQL Enterprise Manager) and select logins. For the login that you are using, alias that login as dbo (stands for database owner) for the databases you are using. That should give you the privileges you need. We also suggest you read the SQL Server documents to understand their security features (they will certainly explain them better than we can).

5.6 Test Procedure

Once the applications have been installed in the destination directory, there are several tests which should be run to verify that the software is working on your system. The section presents the test procedure in order. You may be able to run these tests in a different order but it is safest to initially run them in the order presented.

5.6.1 Examine Images

In this section, you will examine the test images and any images that you already have. The test programs distributed with this software assume that image files consist of a stream of bytes that correspond to the DICOM little-endian (implicit) transfer syntax as defined in Part 5 of the Standard. We also have a switch that allows us to examine images that were stored in big-endian format. We are working on support for DICOM Part 10 files and would appreciate any reports of the test programs failing on Part 10 files.

1. Obtain test images from our ftp site or from the distribution CD. At the ftp site (ftp.erl.wustl.edu), these will be in the directory `/pub/dicom/images/version3/ctntest.` Place the test images in the `images` directory of the distribution.
2. Run the program `dcm_dump_file` on one of the images in the images directory:
   ```
   dcm_dump_file image
   ```
   This program will produce verbose information on the standard output and will print a description of all of the attributes in the image.
3. Run the program `dcm_dump_file` on one or more of the images that your organization has or produces:
   `dcm_dump_file image` or
   `dcm_dump_file -b image`
   The -b switch is used for images that are stored in big-endian byte order. Hopefully, the program will print a similar description of your image(s). If the program fails, try adding the -v switch for even more verbose information. If this still fails, you need to contact the CTN provider to find out why their software does not understand your image format.
4. Run the program `dcm_verify` on one of the supplied test images:
   ```
   dcm_verify image
   ```
   This program examines an image and tries to determine if it includes all of the Modules and Attributes as defined by Part 3 of the Standard. It looks at each Information Entity and dumps information to the standard output. The last part of the output is a summary of type 1 and type 2 attributes that are missing. Hopefully, we have not included any images that are incomplete.
5. Run the program `dcm_verify` on one of your images. Since you have already accomplished step 2 above, you should know if you need the -b switch for this program. You will get a detailed list of the Information Entities, Modules and Attributes present in the image. The program will also print a list of type 1 and type 2 attributes that are missing from your image. If your V3 image fails to pass this test, it will likely cause problems for the rest of our software. You should examine the output of this program and Part 3 and correct your images to include all of the required attributes (or contact the CTN provider who may have incorrectly implemented the rules for this test program).
6. This test is only available on Unix systems.
   Run the program `dcm_x_disp` on one of the test images as well as one of your own images. `dcm_x_disp` will display the image on an X11 display; it should look "reasonable", although there may be problems with window center and width. `dcm_x_disp` understands the -b option. Don't forget to set the `DISPLAY` environment variable. `dcm_x_disp` requires certain parameters to be present before an image can be displayed. See the manual page for `dcm_x_disp` if there are problems.

It is possible that your organization may not agree with the summary information produced by the `dcm_verify` program. If you feel that we have made an error, please contact us so we can correct the problem.

5.6.2 Test Network and Snooper Software

This section describes a procedure for testing network connections between CTN applications and vendor applications. These tests demonstrate that Associations can be established which exercise the storage and verification classes. The last test exercises the DICOM communications monitoring (snooper) software that was specifically developed for the Solaris 2.x environment. The snooper software is only available under Solaris.

1. Run the program `simple_storage` which acts as an SCP of the storage and verification classes:
   ```
   simple_storage portnumber
   ```
   `portnumber` is the TCP/IP port address you choose for this server. It is typically the well known port number reserved for DICOM applications, 104. On Unix systems, you will need to be root to use this port number. You may prefer to choose a different number. For example, we run our tests with port 2100.
   Once the server has started, try to establish a connection with dicom_echo:
   ```
   dicom_echo -c DICOM_STORAGE hostname portnumber
   ```
   The -c switch tells `dicom_echo` to use `DICOM_STORAGE` as the called Application Entity Title. `hostname` is the name of the machine which is running `simple_storage`. `portnumber` is the number you selected for simple_storage at the top of this step.
   If everything works as expected, `dicom_echo` will print several lines of summary information that are extracted from the C-ECHO Response message. The last line should be "Verification Successful". If this does not work as expected, run both `simple_storage` and `dicom_echo` with the -v switch.
2. `simple_storage` has an application title of DICOM_STORAGE. This program is picky and will reject association requests that do not use the proper called application entity title. This feature can be overridden by using the -i switch on `simple_storage` (-i stands for *ignore some incorrect parameters in the Association request*). If you use the -i switch, `simple_storage` has no mechanism for verifying that the caller's title is recognized. Therefore, you should be able to run whatever application you have which implements the Verification class and have it send a C-ECHO request message to `simple_storage`.
3. Run this test with one of the example images provided on our ftp site. Pick an arbitrary directory to work in. We will call that directory A. Create subdirectories in A with names that correspond to modalities as defined by the Standard: MR, CT, US,... In the directory A, run simple_storage as defined above. Use the program `send_image` to establish an Association with `simple_storage` and send one image:
   ```
   send_image <host name> portnumber imagefile
   ```
   `send_image` should print response messages when finished. If the image transmission was successful, there will be an image file in one of the subdirectories you created. (If you run `simple_storage` without the expected subdirectories, `simple_storage` will create them for you.)
4. Repeat test 3 with one of your own images.
5. Use `send_image` to send an image to your SCP of the Storage Class. There are switches to `send_image` which allow you to set the calling and called titles appropriately. Real applications will be more stringent about checking those parameters.
6. [Note: This test only applies to Solaris 2.x systems on which the SNP facility, DULsnoop extension, and DICOM snooper applications have been installed. This software has only been tested with Ethernet networks using the "`/dev/le`" interface. The test assumes that the same type of network and interface will be used.]
   Use the `dcm_snoop` application to monitor communications as described in step 1 of this section. The `dcm_snoop` application has to be run in super-user mode on a third machine that shares the same network as the two running the `simple_storage` and `dicom_echo` programs. The command on the third machine that needs to be given prior to the execution of the `simple_storage` and `dicom_echo` programs is:
   ```
   dcm_snoop /dev/le ppa host1 host2 portnumber 8192 1
   ```
   `ppa` is the number of the interface (generally 0 unless there is more than one Ethernet interface on the machine) where 0 is for /dev/le0, 1 is for /dev/le1, and so on.
   `host1` is the name/IP address of the machine running `dicom_echo`.
   `host2` is the name/IP address of the machine running `simple_storage`.

`portnumber` is the port number used by `simple_storage`.
`8192` is the buffersize used.
`1` is the number of associations to be monitored.
If everything works as expected, the output of the `dcm_snoop` program will show the exchange of the various PDUs of the DICOM association. The association request and accept parameters will be dumped. The DICOM commands will also be dumped.

5.6.3 Test Database Routines

This section describes a procedure for testing the table facility (TBL) which is used as a basis for the database operations in the CTN. The applications are found in the `../apps/tbltest` directory. These applications should be available after the make: `ttunique, ttinsert, ttdelete, ttselect, ttlayout,` and `ttupdate`. These tests will demonstrate that you have the ability to manipulate records contained in the database. It will be helpful to inspect the source to these routines before running each application to get a feeling for the how the applications are designed and what they actually do.

The first step is to create the database tables. The steps for creating the tables under Unix are included in this paragraph. Configure the database with the appropriate temporary tables needed to run these test procedures. This configuration script is CreateTables and may be found in the `../cfg_scripts/sybase` or `cfg_scripts/msql` directory. If you are using a database system other than Sybase or miniSQL these configuration scripts will need to be modified appropriately. The arguments to this script for this test should be `TBLTest TBLTest`. Upon successful completion, this script will create two new tables in the specified database, *TBL_Persons*, and *UniqueNumbers*.

Use the graphical user interface tools provided by Microsoft to create a database: TBLTest. After this step, you need to create database tables. In the Windows environment, we use the *SQL Enterprise Manager*. Run this application and select *tools->SQL Query Tool*. From this tool, you can select *Load SQL Script* and run the script in `cfg_scripts/mssql_server/createtbltesttables.sql`.

1. The success of the table creation can be checked by running the program `ttlayout`. `ttlayout` requests two pieces of in formation, the database name and the table name. In this example, the database name is TBLTest and the table name is TBL_Persons. This routine should yield the following in formation:
```
Column #: 1  Length: 50  Type: String  Name: FNAME
Column #: 2  Length: 50  Type: String  Name: LNAME
Column #: 3  Length:  4  Type: Signed4 Name: AG
Column #: 4  Length:  4  Type: Signed4 Name: ZIP
Column #: 5  Length:  4  Type: Float4  Name: WEIGHT (Float8 for miniSQL)
```
   This is the layout of the TBL_Persons table, and this output verifies that the function TBL_Layout is working as well as the database initialization scripts.
2. Run the test application `ttinsert` next. If it is successful, it will report the simple message: “All Inserts succeeded”. If not, there will be other error messages that need to be addressed.
3. Next, run the test application `ttselect`. This application will produce the following output if successful:
```
In callback: Count is: 1
Field Name: FNAME [7]: JOE
Field Name: LNAME [7]: JONES
Field Name: AGE [4]: 30
Field Name: ZIP [4]: 63100
Field Name: WEIGHT [5]: 150.100

In callback: Count is: 2
Field Name: FNAME [7]: SMELDA
Field Name: LNAME [7]: SMITH
Field Name: AGE [4]: 40
Field Name: ZIP [4]: 63200
Field Name: WEIGHT [5]: 160.200

In callback: Count is: 3
Field Name: FNAME [7]: JOHN
Field Name: LNAME [7]: JONES
Field Name: AGE [4]: 50
Field Name: ZIP [4]: 63300
Field Name: WEIGHT [5]: 170.500

In callback: Count is: 4
Field Name: FNAME [7]: SMITHY
Field Name: LNAME [7]: SMITH
Field Name: AGE [4]: 60
Field Name: ZIP [4]: 63400
Field Name: WEIGHT [5]: 180.250
ALL DONE--Count: 4
```

4. Run the test application `ttupdate` to test the update function. If successful, this application will report the message "Update operation succeeded". Inspect the last name of the record that has an age field equal to 50, and the last name should now be "Woodrow".
5. The delete function is tested with the application `ttdelete`. If successful, it will report "Delete operation succeeded". Inspect the records in the database to ensure that the record containing a zip code of "63100" is gone.
6. The last function to be tested is a very simplistic unique number generator. The application that tests this function is `ttunique`. Running this application should produce the following output:

```
UN1 iteration: 1  count: 1
UN1 iteration: 2  count: 2
UN1 iteration: 3  count: 3
UN1 iteration: 4  count: 4
UN1 iteration: 5  count: 5
UN1 iteration: 6  count: 6
UN1 iteration: 7  count: 7
UN1 iteration: 8  count: 8
UN1 iteration: 9  count: 9
UN2 iteration: 0  count: 0
UN2 iteration: 1  count: 1
UN2 iteration: 2  count: 2
UN2 iteration: 3  count: 3
UN2 iteration: 4  count: 4
UN2 iteration: 5  count: 5
UN2 iteration: 6  count: 6
UN2 iteration: 7  count: 7
UN2 iteration: 8  count: 8
UN2 iteration: 9  count: 9
```

```
UN3 iteration: 0  count: 0
UN3 iteration: 1  count: 1
UN3 iteration: 2  count: 2
UN3 iteration: 3  count: 3
UN3 iteration: 4  count: 4
UN3 iteration: 5  count: 5
UN3 iteration: 6  count: 6
UN3 iteration: 7  count: 7
UN3 iteration: 8  count: 8
UN3 iteration: 9  count: 9
```

5.6.4 Test Queuing Routines

These tests are only available for Unix systems.

This section describes a procedure for testing the queueing mechanism which will be used by the print server display program. In order to test out the queuing mechanism, a pre-existing utility will be used that was originally designed for `ctndisp`. Although this example may not make much sense at first, successful completion will indicate the the GQ facility is operating properly.

1. Use setenv to set the QUEUE_DIRECTORY environment variable to your current directory (e.g. `setenv QUEUE_DIRECTORY ./`). Use the following command to create a new queue of 10 elements where each element is 516 bytes in length (the queue id is 0):
   ```
   gqinitq 0 10 516
   ```
   If no error messages appeared, the command succeeded. You can use the ipcs command to check and make sure that there is one semaphore and one shared memory resource that belongs to your current login.
2. Put some elements on this queue with the enq_ctndisp command. Use the following commands:
   ```
   enq_ctndisp 0 image1 dpn1 1 1
   enq_ctndisp 0 image2 dpn2 2 2
   enq_ctndisp 0 image3 dpn3 3 3
   ```
   If no error messages appeared, the enqueues were successful.
3. Examine the queue with the command `pq_ctndisp 0`. The ouput should look something like the following:
   ```
   <<<  HEAD >>>
   Queue Element: 1
   Image File: image1
   DPN id: dpn1
   Connection: 1--  Image num: 1
         .
         .
         .
   <<< TAIL >>>
   ```
4. Try removing the queue with the command gqkillq command:
   ```
   gqkillq 0 516
   ```
   The command operates silently unless an error occurred. Use the command ipcs to determine that the semaphore and shared memory resources once allocated to your login id are now gone.

5.6.5 Test Demonstration Programs

Once you have completed the tests above, you have demonstrated the pieces that are necessary to run the demonstration programs. There is no detailed procedure to test those programs. The *User's Guide for CTN Demonstration Applications* should provide sufficient information for configuring and running those programs.