CTN User's Guide to Demonstration Applications

Stephen M. Moore
Aniruddha S. Gokhale
David E. Beecher

Mallinckrodt Instiute of Radiology
Electronic Radiology Laboratory
510 South Kingshighway Boulevard
St. Louis, Missouri 63110
Voice: 314-362-6965
Fax: 314-362-6971

Version 2.10.0
August 3, 1998

A CTN contains several demonstration programs that are designed to demonstrate DICOM connectivity. Each demonstration program is designed to highlight a particular use of the DICOM standard. This document contains the user's guide for each demonstration program.

1.0 CTN Configuration

The Central Test Node (CTN) demonstration applications use a relational database (Sybase or miniSQL) to store configuration information as well as data describing images, patients and other objects. The configuration information is stored in a set of tables or relations. Some configuration information is common across a number of demonstrations and will be described in this section. Other configuration information is specific to one application and will be described in later sections of this document.

As noted above, the CTN software uses a relational database product. In our documentation, the term *table* is synonymous with *relation* (as defined by the database community). A *database* is a collection of tables. The CTN will use multiple tables for storing configuration information and data.

1.1 Control Database

The CTN has one control database that stores all of the configuration information for all of the CTN demonstration applications. All CTN demonstration applications use *CTNControl* as the default control database; the default can usually be overwritten with a command-line switch for users that wish to test different configurations.

The *ApplicationEntity* table is used by the CTN to define all application entities that are known by the CTN. This includes applications that execute on the CTN as well as applications that are defined by external systems. Any CTN demonstration application will be defined in the *ApplicationEntity* table. Furthermore, external applications need to be defined in this table to be recognized by CTN demonstration applications. Table 1 below describes the columns in this table. Note that the table has columns for a node name and port number. CTN software only supports the TCP/IP communications stack and has no provisions for the OSI stack. CTN applications can use the values in the *ApplicationEntity* table to make a presentation address that is passed to CTN software to initiate Associations.

The *SecurityMatrix* table contains the list of external applications that are allowed to make connections to a CTN. For each CTN application that acts as a DICOM responder, this table defines the set of applications that are allowed to make a connection. Other tables defined below will indicate what privileges the external applications may have once an Association is established. This table only defines an application's right to establish an Association. This table is not used to regulate Associations *initiated* by a CTN demonstration application. Table 2 below identifies the columns in the *SecurityMatrix* table.

Each row in the *SecurityMatrix* table defines a connection that can be initiated by an external application and accepted by a CTN demonstration application. This means that a popular CTN application (e.g., an image server) will have a number of rows in the *SecurityMatrix* table with different values for RequestingApplication and the same value in the RespondingApplication column.

TABLE 1. Organization of *ApplicationEntity* Table

Column	**Description**
Title	This is the Application Entity Title that identifies an application. It is used as part of Association negotiation as either a "called title" or the "calling title." This title uniquely identifies an application that can exist in only one place on the network. Two applications may not run on different nodes and share the same title.
Node	This is the node name where this application is found. This is the name as found in the UNIX hosts tables for making and receiving network connections. The node name can be a simple name (wuerl) or a fully qualified name (wuerl.wustl.edu). The CTN software does not demand a fully qualified name. Managers may wish to use full domain names for CTNs that will accept connections from other domains.
Port	This field identifies the TCP/IP port number at which a DICOM acceptor is listening for a TCP level connection. The CTN software does not assume the DICOM default TCP/IP port number and requires that this field be entered for DICOM acceptors. The field can be empty for applications that are only initiators of associations.
Comment	This field allows the CTN manager to enter comments which might help to identify the owner of the application and possibly its capabilities. This field is not actively used by the CTN sofotware but can be displayed by configuration software.

TABLE 2. Organization of *SecurityMatrix* Table

Column	**Description**
RequestingApplication	This field lists the Application Title of an application that is requesting an Association with a CTN demonstration application. Requesting Applications are considered to be external applications.
RespondingApplication	This field lists the Application Title of a CTN demonstration application that can accept associations.

Figure 1 shows the general model for CTN server applications that accept requests from external applications. The Application Entity titles are *EXTERNALCLIENT* and *CTNSERVER*. These applcations need to be defined in the *ApplicationEntity* table in the control database. We need an entry in the *SecurityMatrix* table to support connections from *EXTERNALCLIENT* to *CTN-SERVER*. Because *EXTERNALCLIENT* is defined outside of the domain of the CTN, we have no security entries for Associations that it will accept (or reject). Tables 3 and 4 below show examples of entries in the control database to support the model in Figure 1.

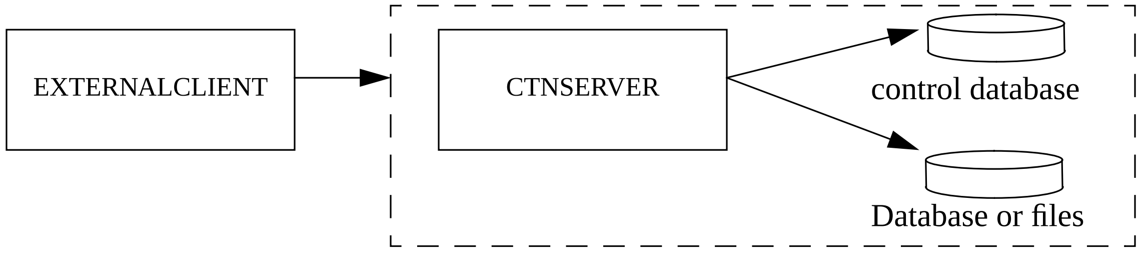


FIGURE 1. General Model for a CTN Server and an External Client

TABLE 3. Example Entries in ApplicationEntity Table

Title	**Node**	**Port**	**Comment**
EXTERNALCLIENT	ws.xyz.com	2100	XYZ whizbang workstation
CTNSERVER	ctn1	104	Image server for Pediatrics

TABLE 4. Example Entries in SecurityMatrix Table

RequestingApplication	**RespondingApplication**
EXTERNALCLLIENT	CTNSERVER

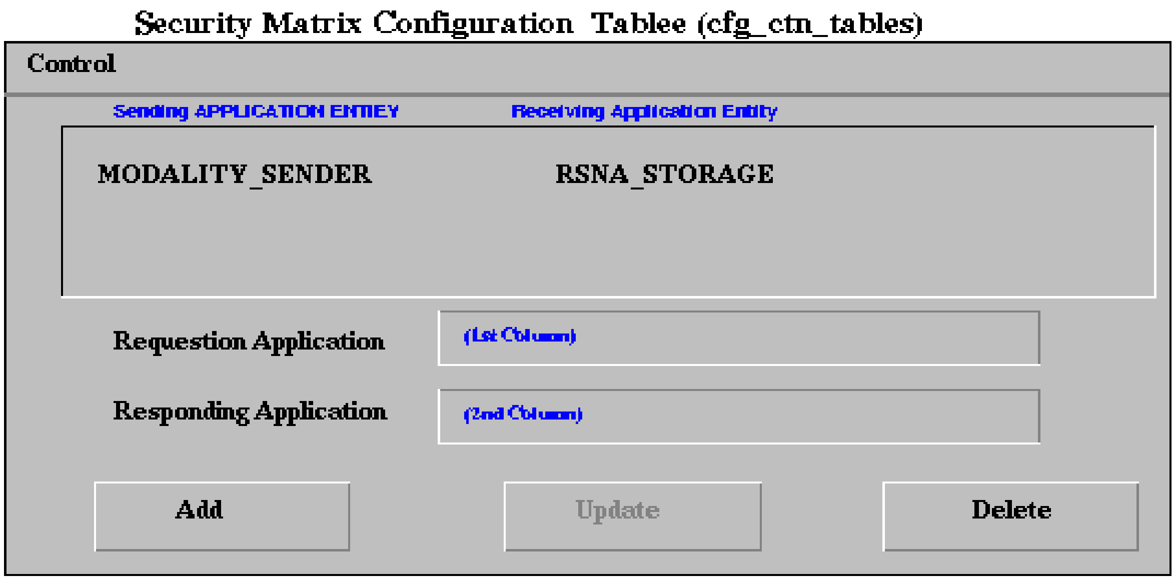
1.2 CTN Master Configuration Tool

This section describes the *cfg_ctn_tables* application. The user can enter configuration information into the *ApplicationEntity* and *SecurityMatrix* tables using one of three methods:

- Enter interactive SQL commands with database product to define entries.
- Use the program *load_control* to read data from a text file into the control database.
- Use the GUI (Motif) presented by *cfg_ctn_tables* to define entries (available under Unix only).

Note: In the paragraph above we referred to interactive SQL as the method for manipulating the database. If the user chooses a different database product, that database product will have a different user interface for data manipulation. It will probably be SQL. We will continue to use such terms as “interactive SQL” which may be particular to Sybase and will assume that readers will make appropriate changes for different databases.

When the user invokes *cfg_ctn_tables*, it displays one window with one pulldown menu. From the pulldown menu, the user can select from a number of different configuration classes. The user should select *General*, which will then allow the user to select *Applications* or *SecurityMatrix*. Selecting an option from the pulldown menu activates an interface that allows the user to modify the entries in one table. Figure 2 below is representative of the displays which are used to manipulate the control tables. We believe that the user interface is simple enough that further explanation is unnecessary with one exception. In many cases, the Update button is shown but unimplemented. Until we implement update, you will need to delete and add entries to perform a logical update.


Security Matrix Configuration Tablee (cfg_ctn_tables)

Control

Sending APPLICATION ENTIEY	Receiving Application Entity
MODALITY_SENDER	RSNA_STORAGE

Requestion Application	(1st Column)
Responding Application	(2nd Column)

FIGURE 2. User Interface for *SecurityMatrix* Table

1.3 Loading Control Tables from ASCII Files

The *cfg_ctn_tables* application is useful for making small changes to the control tables or for viewing the contents of the control tables. We have found it useful to reset the entire control database from a script. In our first effort, we wrote SQL scripts to place data in the Sybase control table. We found the SQL painful and it was difficult to quickly and accurately put together such a script. We wrote a new application to read data from an ASCII file and to load the control tables.

The *load_control* program reads data one line at a time from the standard input.. Each line is one of the following:

- Blank line
- Comment
- Command
- Data

Comments are any line that begins with the character '#'. Commands instruct the program to do something special. Commands begin with a special character and are defined as follows:

'@'	Set the control table. For example '@ ApplicationEntity'
'!'	Set the control table and clear all records from that table. For example, '! Application Entity'.

Data are lines that are to be inserted into a table in the control database. Each data line consists of fields to be inserted separated by the ‘^’ character. This allows fields to contain white space. This version of the software has no support to allow the user to get ‘^’ into the data stream. The user must order the fields according to what the software expects. Table 5 lists the expected order of the fields for each control table. An example input file is found in

cfg_scripts/other/example_load.

TABLE 5. Fields Defined for *load_control* When Loading Control Tables

Table Name	Value 1	Value 2	Value 3	Value 4	Value 5	Value 6
ApplicationEntity	Title	Node	Port	Organization	Comment	
GroupNames	Title	GroupName				
StorageAccess	Access (0)	Title	DbKey	Owner	GroupoName	Comment
StorageControl	RequestingTitle	RespondingTitle	Medium	Root		
SecurityMatrix	RequestingTitle	RespondingTitle				
FISAccess	Access	Title	DbKey	Owner	GroupName	Comment
PrintServerCFG	RequestingTitle	RespondingTitle	GQId			

Table 5 is probably a bit obtuse. Figure 3 is an example of what you will have in an ASCII load file. The entries for the *ApplicationEntity* and *SecurityMatrix* tables should be defined at this point. The other entries will be meaningful after you read the sections on specific demonstration applications.

Note the lines that begin with !, as ! ApplicationEntity. This is a command that means:

- Clear all entries from the *ApplicationEntity* table
- Any data lines that follow are entries to be loaded into the *ApplicationEntity* table.

We continue loading data into the *ApplicationEntity* table until we reach another command (! GroupNames).

```
# example of a file used by load_control
# lines that begin with # are comments
# lines that begin with @ indicate the table to load
# all other lines must contain data, separator is ^,
# blank lines are allowed.
#

! ApplicationEntity
# AE Title     ^ Node      ^ Port ^ Organization ^ Comment
RSNA_Storage   ^ dicomctn  ^ 2100 ^ ERL ^ Q/R SCP for images
RSNAMonochrome ^ dicomctn  ^ 2100 ^ ERL ^ Q/R for monochrome photo IDs
RSNAColor      ^ dicomctn  ^ 2100 ^ ERL ^ Q/R for color photo IDs
PhotoID1       ^ photo     ^ 0    ^ RSNA^ Captures Photo IDs

QueryClient    ^ dicom1    ^ 0    ^ ERL ^ Query SCU, note port = 0
SimpleStorage  ^ dicom1    ^ 2100 ^ ERL ^ Simple storage program
Printer1       ^ p1        ^ 104  ^ XYZ ^ 5th floor printer
Printer2       ^ p2        ^ 104  ^ XYZ ^ Admitting area

! GroupNames
# Title         ^ Group
Printer1        ^ ERL_Printers
Printer2        ^ ERL_Printers

! StorageAccess

# Access ^ SCP Title      ^ Database Key   ^ Owner ^ Group  ^ Comment
0        ^ RSNA_Storage   ^ DicomImage     ^ None  ^ None   ^ Image
0        ^ RSNAMonochrome ^ RSNAMonochrome ^ None  ^ None   ^ Monochrome
0        ^ RSNAColor      ^ RSNAColor      ^ None  ^ None   ^ Color

! StorageControl

# Requesting ^ Responding     ^ Store Medium ^ (File) root
PhotoID1     ^ RSNAMonochrome ^ Medium       ^ /images/video-monochrome
PhotoID1     ^ RSNAColor      ^ Medium       ^ /images/video-color

! SecurityMatrix

# Requesting  ^ Responding
PhotoID1      ^ RSNAMonochrome
PhotoID1      ^ RSNAColor
QueryClient   ^ RSNA_Storage
QueryClient   ^ RSNAMonochrome
QueryClient   ^ RSNAColor
```

FIGURE 3. Example Load File for *load_control*

1.4 Creating Databases and Tables

1.4.1 Unix: Creating Databases and Tables

The CTN software contains a number of scripts for creating the various databases and tables that are required for the demonstration applications. The scripts are found in *cfg_scripts/sybase* or *cfg_scripts/msql*. You may have already used some of the scripts in creating a test database per instructions in the installation manual. Each demonstration program uses the same control database. This is created one time as follows:

CreateDB CTNControl DEV_DICOMIS LOG_DICOMIS 10 5	**(**Sybase)
CreateDB CTNControl	(msql)

This creates the one control database that is needed with a 10 MB data area and a 5 MB log area. Based on your installation of sybase, you may choose different parameters rather than DEV_DICOMIS and LOG_DICOMIS. The process for msql does not specify how large the control table will be. These tables will grow as needed.

Once the control database has been created, the control tables are created as follows:

CreateTables Control CTNControl

This command creates a set of tables of type *Control* in the database named *CTNControl*. Each demonstration program may require different databases for data. These databases and the scripts for creating those databases will be described with each application.

1.4.2 Windows: Creating Databases and Tables

The CTN software supports the Microsoft SQL Server (Version 6.5) database. We create databases using the management tools (GUI) provided with SQL Server. The first step in the process is to use those tools to create a database called *CTNControl*. This can be a small database because it only contains control information. The data/log sections can be 5 and 2 MB.

The second step is to create the tables in the control database. Use the *SQL Enterprise Manager* provided with SQL Server and select *tools->SQL Query Tool*. From this tool, you can select *Load SQL Script* and run the script in

`cfg_scripts/mssql_server/CreateControlTables.sql`.

This script creates the control tables in the *CTNControl* database that you created.

2.0 Image Archive

The image archive is an extension of the *image_server* program which was originally included with the CTN software. This section will discuss the model of the image archive and how the applications are configured and run.

2.1 Image Archive Model

The image archive is a collection of several applications coupled to databases with control, image and HIS/RIS data. Figure 4 shows a model of the image archive.

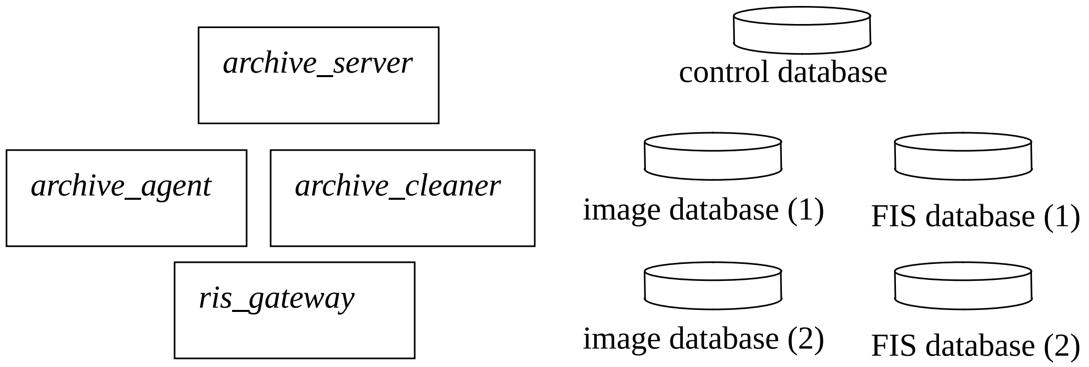


FIGURE 4. Model for Image Archive

The applications *archive_server*, *archive_agent*, *archive_cleaner* and *ris_gateway* collaborate to provide the functions of the image archive. Each image archive has one control database which contains configuration information for the system. As with the *image_server* application, the image archive can appear as several different archive systems. We will use the term *archive system* to describe the collection of applications on the image archive and one set of image and FIS databases. Each archive system would have its own image database and FIS database. An FIS is a Fake Information System, our not quite ready for prime time system for HIS/RIS data. An image archive with several image and FIS databases will have only one control database.

The *archive_server* application accepts DICOM associations from other devices. It supports the following functions:

- Storage of composite objects (images, RT objects)
- Query/retrieve of composite objects (images, RT objects)
- Storage commitment requests

The *archive_server* may initiate DICOM associations to store images to a third party as a result of a C-Move request. The *archive_server* receives the N-Action request which is part of the storage commitment push model SOP class and responds with an N-Action response. The second half of the storage commitment request is handled outside this application.

The user needs to run only one copy of the *archive_server* on an archive machine. The *archive_server* will determine which archive system is being accessed by the Application Entity Title in the DICOM Association Request.

The *archive_agent* application runs in tandem with the *archive_server.* The agent polls a work queue which is filled by the *archive_server* and performs tasks on behalf of the *archive_server.* The user must run a separate copy of the *archive_agent* for each archive system on the machine. This allows the application to poll one work queue and carry out the tasks for a single archive system.

The *archive_agent* supports one set of jobs. It examines storage commitment requests received by the *archive_server* and determines if the image archive has received the images that are requested for storage commitment. The *archive_agent* makes the commitment determination and initiates a separate DICOM association to inform the original application of the status of the commitment request.

The *archive_cleaner* is invoked by the user who wants to delete images from one archive system. The user specifies which system is to be cleaned and a delta time. All images and studies older than this delta time are deleted. This application runs one time and then exits. We envision that users will run this as needed by hand or via an automated script.

There is no automated software that deletes images and studies in response to other events in the system. For example, we do not detect when a database or disk becomes full. Enhancements of this type are left for future releases.

The *ris_gateway* is a server process similar to the *archive_server.* It accepts connections from applications that want to perform some of the HIS/RIS operations that are not directly supported by the *archive_server.* Because this process runs in parallel with the *archive_server,* the AE titles of the *ris_gateway* will differ from those of the *archive_server.* This will make one archive look like two application entities to the outside world. The ris_gateway is included with the image archive to support storage of reports using DICOM HIS/RIS services.

2.2 Services Supported by Image Archive

The document CTN Conformance Statements provides conformance statements for all applications. Table 6 is the “quick list” of services supported by the image archive.

TABLE 6. Servicves Supported by Image Archive

SOP Class UID	SOP Class Name	Role
1.2.840.1008.1.1	Verification	SCP
1.2.840.1008.20.1	Storage Commitment Push Model SOP Class	SCP
1.2.840.1008.3.1.2.5.1	Detached Results Management SOP Class	SCP
1.2.840.1008.3.1.2.5.4	Detached Results Management Meta SOP Class	SCP
1.2.840.1008.3.1.2.6.1	Detached Interpretation Management SOP Class	SCP
1.2.840.1008.5.1.4.1.1.1	Computed Radiolography Image Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.2	CT Image Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.3	Ultrasound Multi-frame Image Storage (Retired)	SCU/SCP
1.2.840.1008.5.1.4.1.1.4	MR Image Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.6	Ultrasound Image Storage (Retired)	SCU/SCP
1.2.840.1008.5.1.4.1.1.7	Secondary Capture Image Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.12.1	X-Ray Angiographic Image Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.12.2	X-Ray Radiofluoroscoping Image Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.12.3	X-Ray Angiographic Bi-Plane Image Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.20	Nuclear Medicine Image Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.481.1	RT Image Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.481.2	RT Dose Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.481.3	RT Structure Set Storage	SCU/SCP
1.2.840.1008.5.1.4.1.1.481.5	RT Plan Storage	SCU/SCP
1.2.840.1008.5.1.4.1.2.1.1	Patient Root Query/Retrieve Information Model - FIND	SCP
1.2.840.1008.5.1.4.1.2.1.2	Patient Root Query/Retrieve Information Model - MOVE	SCP
1.2.840.1008.5.1.4.1.2.1.3	Patinet Root Query/Retireve Information Model - GET	SCP
1.2.840.1008.5.1.4.1.2.2.1	Study Root Query/Retrieve Information Model - FIND	SCP
1.2.840.1008.5.1.4.1.2.2.2	Study Root Query/Retrieve Information Model - MOVE	SCP
1.2.840.1008.5.1.4.1.2.2.3	Study Root Query/Retrieve Information Model - GET	SCP
1.2.840.1008.5.1.4.1.2.3.1	Patient/Study Only Query/Retrieve Information Model - FIND	SCP
1.2.840.1008.5.1.4.1.2.3.2	Patient/Study Only Query/Retrieve Information Model - MOVE	SCP
1.2.840.1008.5.1.4.1.2.3.3	Patient/Study Only Query/Retrieve Information Model - GET	SCP

2.3 Image Archive Control Tables

This section describes the configuration tables used by the archive server in addition to the *ApplicationEntity* and *SecurityMatrix* tables. Readers who are familiar with the tables may want to skip to the summary in Section 2.3.1.

The *StorageAccess* table is used to control access to storage areas by providing a mapping from an Application Entity Title to a storage area with access privileges. When the *archive_server* receives an Association request, it uses the Called Application Title as the key for this table. This application title will identify one row in the *StorageAccess* table which defines the storage area and privileges extended to the calling application. Table 7 below describes the columns in the *StorageAccess* table.

TABLE 7. Organization of *StorageAccess* Table

Column	Description
Title	This field lists the Application Title of an *archive_server* application that is running on a CTN. The value in this field must be unique and will be used as the key to search this table
DbKey	This field contains a key value that will be used to open the database that contains the tables which implement the DICOM hierarchical query model (for query/retrieve). In the CTN implementation, the key value is actually the "name" of the database.
Owner	This field defines an application that is the owner of this particular storage area. The owner is defined as the Application Entity Title of an application. This field can be null.
GroupName	This field identifies a group name that is assigned to the storage area. A group name is an arbitrary name defined by the CTN administrator and will *not* be an Application Entity Title. This field can be null.
Access	This field contains an integer value which is used to allow access privileges for the owner, group and world (other) applications. Current privileges that are identified are read and write. This field cannot be null (but is not used in this release); enter 0.
Comment	This comment field allows the CTN administrator to enter a comment which could describe the storage area.

Storage areas are similar to UNIX file systems in that they have one owner and are also labeled with a group. The owner is one external application that is defined by the CTN administrator. The group is an arbitrary name that will be used to collect a series of external applications. We identify an owner and a group so that we can extend special access privileges to the owner. For example, we might let the owner store images in the storage area and let members of the group read images from the storage area. The owner and group privileges are unimplemented features in this release. The user can enter arbitrary values in those fields. Figure 3 shows example values that can be loaded using the *load_control* application.

The *StorageControl* table is used to define where images will be stored by the image server when it receives images. The table is indexed by both the particular image archive and the application title of the requesting application.. This allows the image archive to maintain separate directories for image storage in the same area. Table 8 below describes the columns in the *StorageControl* table.

TABLE 8. Organization of *StorageControl* Table

Column	Description
RequestingApplication	This field contains the title of an application that is requesting to store images on the image archive.
RespondingApplication	This field contains the Application Entity Title of a CTN image archive.
Medium	This field is included for future applications.
Root	This field identifies a root path where images will be stored for this combination of requesting application and responding application. This is a UNIX absolute path name which identifies a directory.

Please refer to the example in Figure 3. The first entry under StorageControl means that images sent from *PhotoID1* to *RSNAMonochrome* will be stored in */images/video-monochrom*e.

The *FISAccess* table is similar to the *StorageAccess* table. This table maps the Application Entity title of CTN applications to the name of FIS databases on the CTN. The columns and usage are the same as for the *StorageAccess* table shown in Table 7.

2.3.1 Summary of Image Archive Control Tables

Table 9 is a brief summary of all of the control tables used by the image archive.

TABLE 9. Summary of Control Tables Used by Image Archive

Table	Required Entries
ApplicationEntity	One entry for each archive system. One entry for each device known by the archive systems.
SecurityMatrix	One entry for each device that wants to connect to an image archive system. If a device is to connect to more than one system in an archive, an entry is required for each connection.
StorageAccess	One entry for one archive system. This table will map the AE title of the archive system to the name of the image database.
StorageControl	One entry for each device that sends images to one archive system. This table maps the sending device name and archive system name to a directory where the archive system stores images. If a device will send images to N archive systems, N entries are required in this table.
FISAccess	One entry for one archive system. This table will map the AE title of the archive system to the name of the FIS database.

2.4 Creating Image Archive Data Tables

2.4.1 Unix: Creating Image Archive Data Tables

Each archive system requires one image database and one FIS database. The CTN software provides scripts (*cfg_scripts/sybase, cfg_scripts/msql*) for creating the databases and tables. Our normal procedure is to create two databases; we place the image tables in one database and the FIS tables in the other database. Example commands for creating these tables (in sybase) are:

```
CreateDB DicomImage DEV_DICOMIS LOG_DICOMIS 100 25
CreateTables DIM DicomImage
CreateDB FISTable DEV_DICOMIS LOG_DICOMIS 10 5
CreateTables FIS FISDB
```

The scripts for msql are similar:

```
CreateDB DicomImage
CreateTables DIM DicomImage
CreateDB FISDB
CreateTables FIS FISDB
```

Section 2.6 below will have suggestions for database names for testing.

2.4.2 Windows: Creating Image Archive Data Tables

The Windows version of the image archive does not support the FIS database, but it does support the image database. Use the SQL Server management tool to create a database called *DicomImage.* After the database has been created, you need to create database tables. Use the *SQL Enterprise Manager* provided with SQL Server and select *tools->SQL Query Tool.* From this tool, you can select *Load SQL Script* and run the script in

cfg_scripts/mssql_server/CreateDIM2Tables.sql

This will create the image tables in the *DicomImage* database. If you want to change the database name, you need to create a different database in the first step and edit the `CreateDIM2Tables` sql script to use the proper database.

2.5 Invoking Image Archive Applications

The image archive uses several applications described above. The normal pattern is to invoke one copy of *archive_server* and one copy of *ris_gateway* for an image archive. Each archive can respond as multiple archive systems, and the user will invoke one copy of *archive_agent* for each archive system. These processes run continuously as server processes. The user will invoke *archive_cleaner* on an as needed basis as the disk system or database become full for each archive system.

The *archive_server* application is invoked as follows:

archive_server [-e] [-f db] [-i] [-l logfile] [-n node] [-o max] [-q] [-r] [-t] [-v] port

-e	Examine received images and perform SOP validation. If the image does not pass SOP tests, return a failure code to sending app.
-f db	Use *db* as control database rather than default (CTNControl)
-i	Ignore some tests during association validation. Effectively turns off the use of the Security Matrix.
-l logfile	Log association requests to *logfile.*
-n node	Use *node* as the name of the system rather than using hostname.
-o max	Allow max simultaneous connections from an organization. (Used in RSNA demonstrations.)
-q	Quiet mode; don’t dump a lot of messages to terminal.
-r	Reduced capability. Do not open FIS tables. This is useful for those systems that do not care about FIS functions, like storage commitment. It is a required switch in the Windows verson.
-t	Use threads rather than forking a new process. Required under Windows.
-v	Place DUL and SRV facilities in verbose mode.
port	TCP/IP port address. No default.

The Windows version requires the -r and -t switches. The FIS part is not implemented yet under windows. These are optional in the Unix implementation.

The *ris_gateway* application is invoked as follows:

ris_gateway [-f db] [-i] [-n node] port

-f	Use db as control database instead of default (CTNControl)
-i	Ignore some tests during association validation. Effectively turn off the use of the Security Matrix.
-n node	Use *node* as the name of the system rather than using hostname.
port	TCP/IP port address. No default.

The *archive_agent* application is invoked as follows:

archive_agent [-d db] [-f control] [-h] [-i fis]

-d db	Set image database name (default LTA_IDB).
-f control	Set control database name (default CTNControl).
-h	Print help message and exit.
-i fis	Set FIS database name (default LTA_FIS)

Note there is no option for setting an AE title. The *archive_agent* picks this up from work entries found in the FIS database.

The *archive_cleaner* application is invoked as follows:

archive_cleaner [-d dbname] [-h] [-n] days [hours [minutes]]

-d dbname	Set image database name (default LTA_IDB).
-h	Print help message and exit.
-n	Do not actually perform the delete.
days, hours, minutes	Delete studies older than the sum of these times. User is only required to specify days, but may specify hours and minutes as well.

We commonly delete “new” studies during testing by specifying (days, hours, minutes) of (0, 0, 30). The *archive_cleaner* takes the current time of day, subtracts the time delta entered by the user and deletes studies that were acquired before the cutoff time. The *archive_cleaner* then takes a second pass through the system and deletes any patient records with no studes that are older than the cutoff time.

The WIndows software only supports the `archive_server` in this release. Future releases will include more functions.

Windows pplications 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
```

For example, you might set this to ARCHIVE:sa:password. 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.

2.6 Image Archive Setup and Test Procedure

This section describes a test configuration and procedure for testing the image archive. The system will support four application entities which implement a long term archive and a short term archive. These two archive systems (long term archive, short term archive) will each have two Application Entity titles.

The AE titles for the archives will be CTN_LTA (for long term archive), CTN_LTA_FIS, CTN_STA (for short term archive), and CTN_STA_FIS. Table 10 describes the databases required to support this test configuration.

TABLE 10. Databases Required to Support Two Archive Systems on one Image Archive

AE Title	Database Name	Table Type
	CTNControl	Control
CTN_LTA	LTA_IDB	Image (DIM)
	LTA_FIS	Information System (FIS)
CTN_LTA_FIS	LTA_FIS	Information System (FIS)
CTN_STA	STA_IDB	Image (DIM)
	STA_FIS	Information System (FIS)
CTN_STA_FIS	STA_FIS	Information System (FIS)

Follow these steps to configure and test the image archive. This procedure will test the image storage and query/retrieve functions of the server. The modality emulator test procedure below will be used to verify storage commitment. The report workstation test procedure below will be used to verify some of the HIS/RIS operations of the *ris_gateway*.

1. Create the set of control tables using the scripts provided. Use *CTNControl* for the name of the control database.
2. Create two image databases and two FIS databases using the names listed in Table 10 above. There are scripts provided in *cfg_scripts/sybase* and *cfg_scripts/msql*. In the Windows environment, you will need to create the image databases using the SQL Server tools and create the image tables by modifying the script in *cfg_scripts/mssql_server/CreateDIM2Tables.sql*.
3. Create entries in the *ApplicationEntity* table for two image archive servers:
   CTN_LTA
   CTN_STA
   These should run on one machine and can respond to the same port number.
4. Create entries in the ApplicationEntity table for two RIS gateway servers (Unix only):
   CTN_LTA_FIS
   CTN_STA_FIS
   These should run on the same machine as the archive server processes but will need to use a different port number.

5. Create an entry in the *ApplicationEntity* table for one application that will communicate with the image archive. Use the name **CTN_INSTRUMENT** for this application. It can be defined on the same machine as the archive or on a different machine. You will need to define a TCP/IP port number for this application. If you plan to run this application on the same machine as the image archive applications, select a port number which is different from the port number used by the archive systems.
6. Add two entries in the *SecurityMatrix* table which specify that **CTN_INSTRUMENT** can connect to **CTN_LTA** and **CTN_STA**. You do not need entries which allow the image archive applications to connect to the **CTN_INSTRUMENT**.
7. Create entries in the *StorageAccess* and *FISAccess* tables which will map the Application Entity titles of the archive systems to particular databases. Use the mappings provided in Table 10 above. For WIndows, ignore the mappings for *FISAccess*.
8. Ignore the owner and access privileges in the *StorageAccess* and *FISAccess* tables. Ignore the *GroupNames* table.
9. Create two entries in the *StorageControl* table that tells the image archive where to store images that are sent to it. For example, you might specify that images from **CTN_INSTRUMENT** to **CTN_LTA** would be stored on */images/lta* while images from **CTN_INSTRUMENT** to **CTN_STA** would be stored on */images/sta*. The *archive_server* will create the paths if they do not already exist on your system.
10. Run the *archive_server* application. For windows. you must use the -r and -t switches.
11. Send images to the two archive systems:

    send_image -a CTN_INSTRUMENT -c CTN_LTA host port image [image]
    send_image -a CTN_INSTRUMENT -c CTN_STA host port image [image]
12. Use the *query_client* program to query the archive_server about the images you transmitted. The *query_client* application allows you to fill in both the called and calling AE titles. For the calling AE title, use **CTN_INSTRUMENT**. For the called AE title, use **CTN_LTA** or **CTN_STA**.
13. Run the *simple_storage* application as a server process for **CTN_INSTRUMENT**. Turn off its security tests by using the -i option. The port number used should match the port number you defined for **CTN_INSTRUMENT** in Step 4.
14. Direct the *query_client* application to move a study to **CTN_INSTRUMENT**. The *archive_server* should establish a connection with *simple_storage* and transmit the images in the study.
15. Send bug reports or requests for help to *dicom_bugs@wuerl.wustl.edu*.

3.0 Modality Emulator

The Modality Emulator is a collection of applications which are intended to act as a modality that communicates using the DICOM protocol. These applications support several SOP classes that one would expect to find in a modality. The first version is rather crude. Later versions are expected to have configuration tools to allow you to specify the type of modality and what services are supported.

This is supported in the Unix versions of the software but not the Windows version.

3.1 Modality Emulator Model

The modality emulator consists of several applications and CTN control, image and FIS databases. Figure 5 shows the model of the emulator with one set of image and FIS databases. With proper configuration, a user can create several logical modality emulators on one machine.

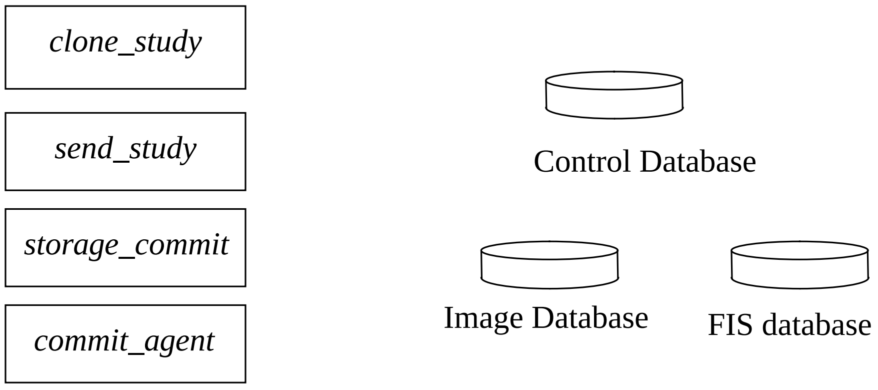


FIGURE 5. Model for Modality Emulator

The modality emulator keeps a list of images in the Image Database using the normal CTN database software. Each image in the database will have a pointer to a file located on the emulator. We have implemented a mechanism that allows us to reuse the image files on the emulator. The result is that the Image Database appears to have a greater number of patients and studies without filling up the disk with image files.

The *clone_study* application creates new study entries in the Image Database by cloning existing studies. The user specifies which study to clone from command line parameters. The *clone_study* application retrieves the database entries for each image in the original study and creates entries for a new study. The application updates the identifying fields in the database so that the new images belong to a different patient and study. When the entries are written back into the database, the path to the image files remain unchanged. This allows us to create new studies without duplicating all of the image files.

The *send_study* application sends one or more studies that are defined in the image database to an application which provides DICOM storage. This application is driven by command line options that designate the studies to be sent and the destination. The user may send all studies for one study or all studies for one patient. The *send_study* application exits after it has sent the requested studies to the receiving application. This application reads information from the control and image databases, but makes no record of what has been transmitted. The *send_study* application normally transmits the contents of the image files associated with the study. Because the *clone_study* application can place cloned studies in the database with pointers to the original images, *send_study* has an option that allows the user to override the identifying information in the image files with information from the database.

The *storage_commit* application sends a storage commit request for a set of images designated by the caller. It uses the same command line switches as *send_study* to allow the user to request storage commitment for the images in one study or the images for one patient. The *storage_commit* application reads the image database to determine for which images it should request storage commitment. The application makes a set of entries in the FIS database which logs that a request has been made and transmits the storage request. After the N-Action response to the storage commitment request has been received, the *storage_commit* application releases the DICOM association and assumes the storage commitment SCP will initiate a separate association to return the results of the storage commitment request. This implies that there will a problem if the storage commitment SCP wants to send the storage commit event report on the original DICOM association.

The *commit_agent* application is a server process that receives DICOM associations and processes N-Event requests for the Storage Commitment Push Model SOP Class. This is the companion program to the *storage_commit* application that is waiting for the response message from the Storage Commitment SCP indicating the status of the commit request. This application processes the response and updates the FIS tables which recorded the initial request. It updates the time the response was received, but takes no other action. That is, this application will print the number of SOP instances that were successfully committed and the number that were not successfully committed, but does not make any additional database entries.

3.2 Modality Emulator Control Tables

The modality emulator uses the same control tables used by the image archive. Table 11 lists the control tables used and summarizes the required entries.

TABLE 11. Control Tables Used by Modality Emulator

Table	**Required Entries**
ApplicationEntity	One entry for the modality emulator. One entry for each device known by the modality emulator.
SecurityMatrix	One entry for each device that will receive storage commitment requests from the modality emulator. This table is used to allow the DICOM associations from those devices with the results of the commit request.
StorageAccess	One entry for one modality emulator. This table will map the AE title of the modality emulator to the name of the image database.
StorageControl	No entries needed. Modality emulator does not receive images from other devices.
FISAccess	One entry for one modality emulator. This table will map the AE title of the modality emulator to the name of the FIS database.

3.3 Creating Modality Emulator Data Tables

Data tables for the modality emulator are created using the same procedure documented for the image archive. Each modality emulator requires one image database and one FIS database. The CTN software provides scripts (*cfg_scripts/sybase*, *cfg_scripts/msql*) for creating the databases and the tables in the databases. Our normal procedure is to create two databases; we place the image tables in one database and the FIS tables in the other database. Example commands for creating these tables (in sybase) are:

```
CreateDB DicomImage DEV_DICOMIS LOG_DICOMIS 100 25
CreateTables DIM DicomImage
CreateDB FISTable DEV_DICOMIS LOG_DICOMIS 10 5
CreateTables FIS FISDB
```

The scripts for msql are similar:

```
CreateDB DicomImage
CreateTables DIM DicomImage
CreateDB FISTable
CreateTables FIS FISDB
```

Section 3.5 below will have suggestions for database and table names for testing a specific configuration.

3.4 Invoking Modality Emulator Applications

The modality emulator uses several applications described above. The user needs to invoke one or more applications via command line options to obtain the desired results. There is no GUI that pulls these applications/functions together.

The *send_study* application is invoked as follows:

```
send_study [-a title] [-d dbname] [-f control] [-n] [-p <pat id>]
                [-r <req number] [-u <study UID>] destinationAETitle
```

-a title	Set calling AE title (default CTN_INSTRUMENT)
-d dbname	Set image database name (default INSTRUMENT_IDB)
-f control	Set control database name (default CTNControl)
-n	Make new header in images from database entries
-p <pat id>	Send study (or studies) for patient with patient ID <pat ID>
-r <req num>	Send study with accession number <req num>
-u <study UID>	Send study with study instance UID <study UID>
destinationAETitle	AE title of the application which has a storage SCP. This application must be defined in the control database to obtain a mapping to a host name and port number.

The user must use one switch from the -p, -r -u group to designate the studies to be transmitted. The user cannot select more than one switch from this group.

The -n switch allows the user to override some of the header information that is stored in the image files with information that is found in the database. This is used in conjunction with the *clone_study* application. We can use the same image file for multiple studies, overriding the header information to get the proper identifying information in the image.

The *storage_commit* application is invoked as follows:

storage_commit [-a title] [-d dbname] [-f control] [-i <fis name>] [-p <pat ID>] [-r <req num] [-u <study UID>] destinationAETitle

-a title	Set calling AE title (default CTN_INSTRUMENT)
-d dbname	Set image database name (default INSTRUMENT_IDB)
-f control	Set control database name (default CTNControl)
-i <fis name>	Set the FIS database name (default INSTRUMENT_FIS)
-p <pat ID>	Send storage commit req for images for patient with patient ID <pat ID>
-r <req num>	Send storage commit req for images with accession number <req num>
-u <study UID>	Send storage commit req for images with study instance UID <study UID>
destinationAETitle	AE title of the application which has an SCP of the Storage Commitment Push Model SOP Class. This application must be defined in the control database to obtain a mapping to a host name and port number.

As with *send_study*, the user must specify exactly one of the -p, -r or -u options.

The *clone_study* application is invoked as follows:

clone_study [-d dbname] [-f control] [-p <pat id>] [-r <req number] [-u <study UID>]

-d dbname	Set image database name (default INSTRUMENT_IDB)
-f control	Set control database name (default CTNControl)
-p <pat id>	Send study (or studies) for patient with patient ID <pat ID>
-r <req num>	Send study with accession number <req num>
-u <study UID>	Send study with study instance UID <study UID>

The *clone_study* application creates new identifying information using the UID facility. The user must have a valid uidfile which is named by the environment variable UIDFILE.

The *commit_agent* application is invoked as follows:

commit_agent [-f db] [-i] [-l logfile] [-n node] [-q] [-v] [-x <fac>] port

-f db	Set control database (default CTNControl)
-i	Ignore some tests during association validation. Effectively turns off the use of the Security Matrix.
-l logfile	Place log of association requests in *logfile*. Do not use the same physical *logfile* that is used for *archive_server.*
-n node	Use *node* as the name of the system rather than using hostname
-q	Quiet mode, don't dump a lot of messages to terminal
-v	Place DUL facility in verbose mode
-x <fac>	Place <fac> in verbose mode (<fac> = TBL, SRV)
port	TCP/IP port number. No default.

3.5 Modality Emulator Test Setup and Test Procedure

This section describes a setup and test procedure for the modality emulator. It requires the operation of the CTN image archive and assumes the configuration steps described in section 2.6 of this document have been completed. The tests described in this section will also verify the Storage Commitment functions provided by the image archive applications. The configuration steps apply to the machine designated as the modality emulator.

1. Add entries to the *SecurityMatrix* table which specify that **CTN_LTA** and **CTN_STA** can connect to **CTN_INSTRUMENT**. This is in addition to the entries above that allow **CTN_INSTRUMENT** to connect to **CTN_LTA** and **CTN_STA**.
2. Create a new image database (INSTRUMENT_IDB) and a new FIS database (INSTRUMENT_FIS) to be used by the modality emulator.
3. Create entries in the *StorageAccess* and *FISAccess* tables which will map the Application Entity title **CTN_INSTRUMENT** to the image database INSTRUMENT_IDB and the FIS database INSTRUMENT_FIS.
4. Ignore the owner and access privileges in the *StorageAccess* and *FISAccess* tables. Ignore the *GroupNames* table.
5. Load a set of studies into the image database INSTRUMENT_IDB using the *fillImageDB* application or *fillImageDBScript*. These are discussed in section 7.1 of this document. This database represents a set of studies that were completed on this modality. We will ignore for now the fact that the header information will indicate different manufacturers' instruments and different modalities.
6. Run the image archive by starting the *archive_server* and *archive_agent* applications. You will need to start one copy of the *archive_agent* for each archive system. We will test with just one archive system, **CTN_LTA**. That means you can use the defaults on *archive_agent*.

7. Send two studies from the modality emulator to **CTN_LTA**:

    ```
    send_study -a CTN_INSTRUMENT -u 1.2....... CTN_LTA
    send_study -a CTN_INSTRUMENT -u 1.2....... CTN_LTA
    ```

    You will have to know the study instance UID of a study in your modality emulator database. You could also use the patient ID switch or req num/accession number switch if you find that more convenient.

8. Run the *commit_agent* for the **CTN_INSTRUMENT**. This will listen for storage commitment results from the image archive:

    ```
    commit_agent port
    ```

9. Send a storage commitment request from **CTN_INSTRUMENT** to **CTN_LTA**:

    ```
    storage_commit -a CTN_INSTRUMENT -u 1.2... CTN_LTA
    ```

    You should use the same study designation that you chose in step 7 to select a study.

10. Examine the results printed by the *commit_agent* application. It should receive a response from the image archive. The response may not be immediate as the *archive_agent* application sleeps for 15 seconds between iterations.

11. Examine the storage commitment requests in the FIS for both the image archive and the modality emulator. This is done with the *dump_commit_requests* command. It will print the date/time the request was made and the date/time the request was serviced. These are stored in the FIS database.

    ```
    dump_commit_requests INSTRUMENT_FIS
    dump_commit_requests LTA_FIS
    ```

12. If you want to be fancy, don't run the *archive_agent* when you run the *archive_server.* The *archive_server* will queue up a work record after it receives the storage commitment request. You can examine the FIS databases in this state. Start the *archive_agent* later and wait for the final storage commitment report to be sent to **CTN_INSTRUMENT**.

13. Send bug reports or requests for help to *dicom_bugs@wuerl.wustl.edu*.

4.0 Results Reporting

The Results Reporting System should be a workstation that allows one to review images and generate diagnostic reports. At this time, this sytem is a collection of applications that allows one to enter results and interpretations into an FIS and send messages to a remote system.

This is supported in the Unix versions of the software but not the Windows version.

4.1 Results Reporting Model

The Results Reporting System consists of several applications and CTN Control, Image and FIS databases. Figure 6 shows the model of the system.

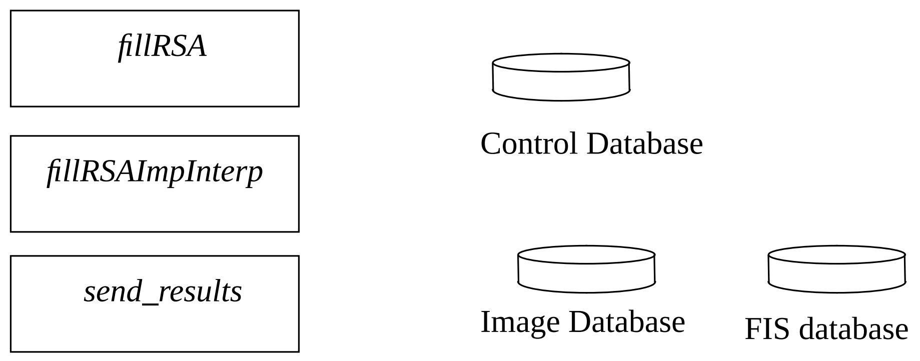


FIGURE 6. Model for Results Reporting System

The *fillRSA* application creates a set of entries in the FIS database from the entries found in the Image database. This application cycles through the Image database and creates corresponding entries in the FIS database for Patient, Study, Study Component, Results and Interpretation objects. This information is used for the results management aspects of this system but could have uses for other applications (e.g., study component information). *fillRSA* create empty entries in the Results and Interpretation objects in the FIS database.

The *fillRSAImpInterp* application reads from the standard input and fills data into the Results and Interpretation objects in the FIS database. This application expects data in a specific format (as a result of history in how the original text files were generated):

vendor
Patient Name
Accession Number
Study Instance UID
1 or more lines of impressions (results object)
****** (1 line with at least 4 *'s)
1 or more lines of Interpretation Text (interpretation object)
****** (1 line with at least 4 *'s)

fillRSAImpInterp reads data in the format described above from the standard input and updates the appropriate records in the Results and Interpretation Tables in the FIS database. An example of a file that we use to fill the database is found in:

apps/fillRSA/DICOM_35_reports.txt

This application takes the second batch of text and places it in both the Interpretation text and Interpretation Diagnosis Description attributes in the Interpretation table.

The *send_results* application reads results and interpretation information from the FIS and sends notification events to a remote system. It specifically sends the events listed in Table 4 below:

TABLE 12. Event Notifications Sent by *send_results*

SOP Class	**Event Name**	**Event ID**
Detached Results Management	Results Created	1
Detached Interpretation Management	Interpretation Created	1
Detached Interpretation Management	Interpretation Approved	4

It is assumed that the "text" of the report is transmitted in the Interpretation Diagnosis Description (4008, 0115). The Results Created Event has the referenced Study Instance UID to tie the results to a study. The Interpretation Created Event contains the referenced Results UID to tie the interpretation to the proper results object (which is in turn tied to the study). This application skips the intermediate steps of Interpretation Recorded and Interpretation Transcribed.

4.2 Results Reporting Control Table

The Results Reporting System uses fewer control tables than most of the CTN applications. Table 13 lists the tables in the control database that are used by the Results Reporting System.

TABLE 13. Control Tables Used by Results Reporting System

Table	**Required Entries**
ApplicationEntity	One entry for each device known by the Results Reporting System.
SecurityMatrix	No entries needed. Reporting system does not receive associations.
StorageAccess	No entries needed.
StorageControl	No entries needed.
GroupNames	No entries needed.
FISAccess	No entries needed.

4.3 Creating Results Reporting System Data Tables

Data tables for the Results Reporting System are created using the same procedure documented for the image archive. Each Results Reporting System requires one image database and one FIS database. The CTN software provides scripts (*cfg_scripts/sybase, cfg_scripts/msql*) for creating the databases and the tables in the databases. Our normal procedure is to create two databases; we place the image tables in one database and the FIS tables in the other database. Example commands for creating these tables (in sybase) are:

CreateDB DicomImage DEV_DICOMIS LOG_DICOMIS 100 25
CreateTables DIM DicomImage
CreateDB FISTable DEV_DICOMIS LOG_DICOMIS 10 5
CreateTables FIS FISDB

The scripts for msql are similar:

CreateDB DicomImage
CreateTables DIM DicomImage
CreateDB FISTable
CreateTables FIS FISDB

Section 4.5 below will have suggestions for database and table names for testing a specific configuration.

4.4 Invoking Results Reporting Applications

The Results Reporting System uses several applications described above. The user needs to invoke one or more applications via command line options to obtain the desired results. There is no GUI that pulls these applications/functions together.

The *fillRSA* application is invoked as follows:

fillRSA [-r title] [-x fac] image-db fis-db

-r title	Enter *title* as retrieve AE title at series level
-x fac	Place *fac* in verbose mode (TBL)
image-db	Database name of image database from which to obtain patients
fis-db	Database name of FIS database to be loaded

This application creates new entries in the FIS database and needs to generate UIDs to do so. The user must have a legal UID file (see *Programmer's Guide to the UID Facility*) and set the environment variable *UIDFILE* to the name (full or relative path) of that file. Example uid files come with the software in the *uids* directory.

The *fillRSAImpInterp* application is invoked as follows:

fillRSAImpInterp [-x fac] FIS-database < text-file

-x fac	Place *fac* in verbose mode (TBL)
FIS-database	The FIS database to be modified
text-file	The file with text which has results and interpretations

The *send_results* application is invoked as follows:

send_results [-a title] [-d dbname] [-f control] [-i <FIS db>] [-r <req num>]
[-u <study UID>] destinationAETitle

-a	Set calling AE title (default CTN_WKSTN)
-d	Set image database name (default WKSTN_IDB)
-f	Set control database name (default CTNControl)
-i	Set FIS database name (default WKSTN_FIS)
-r	Send results from study with accession number <req num>
-u	Send results from study with study instance UID <study UID>
destinationAETitle	AE title of the application which has an SCU of the Results Management and Interpretation Management classes. This application must be defined in the control database to obtain a mapping to a host name and port number.

4.5 Results Reporting Setup and Test Procedure

Steps 1-5 below refer to configuration of the Results Workstaiton. Steps 6-11 refer to configuration of the remote system. We assume the remote system is a CTN image archive.

1. Create an Image database and an FIS database for a workstation. Use the sybase or msql scripts to create WKSTN_IDB and WKSTN_FIS databases.
2. Create the proper tables in the WKSTN_IDB and WKSTN_FIS databases:
   CreateTables DIM WKSTN_IDB
   CreateTables FIS WKSTN_FIS
3. Fill the image database with a set of images. Since we have reports for the RSNA image set, we suggest you use those images:
   fillImageDBScript /sw1/projects/dinpacs/images/rsna95-trim WKSTN_IDB
4. Create entries in FIS based on entries in image database:
   setenv UIDFILE <your uid file>
   fillRSA WKSTN_IDB WKSTN_FIS
5. Fill report text into FIS:
   fillRSAImpInterp WKSTN_FIS < DICOM_35_reports.txt
6. Add an entry in the control database for the remote system which will receive the results. In our testing, we name that system **CTN_LTA_FIS**. It will need to have a different port number than **CTN_LTA**.
7. Configure remote system to receive event reports. In the control database of the remote system, add entries for **CTN_LTA_FIS** and **CTN_WKSTN**.
8. Create entries in *FISAccess* table of remote system to map the AE title **CTN_LTA_FIS** to the FIS **LTA_FIS**.
9. As usual, ignore the owner and access privileges in the *FISAccess* table. Ignore the *GroupNames* table.

10. Add an entry in the *SecurityMatrix* on the remote system to allow **CTN_WKSTN** to connect to **CTN_LTA_FIS**.

11. Start the *ris_gateway* on the remote system to receive event reports. Use a log level (-l switch) of 1 or 2 to see events. A level of 1 gives reasonable information. A level of 2 gives more debugging information and dumps the exact messages sent by the client application.

    ris_gateway -l 1 port

12. Send results to the remote system:

    send_results -r GE0005 CTN_LTA_FIS

Send bug reports or requests for help to *dicom_bugs@wuerl.wustl.edu*.

5.0 Print Manager

The CTN contains a print manager program, *pmgr_motif*, which is designed to demonstrate the print management service provided by the requesting SCU.

This is supported in the Unix versions of the software but not the Windows version.

Based on the Motif Widget set, the *pmgr_motif* application will provide graphical user interfaces for:

1. Selecting printers.
2. Creating a Basic Film Session.
3. Creating a Basic Film Box.
4. Selecting studies from a database.
5. Previewing images belonging to a study.
6. Selecting from different studies the images to print.
7. Set the Basic Image Attributes specific to a selected image.
8. Controlling positions of images appearing on the print film.

The following figure represents the operations and connections of the *pmgr_motif* program.

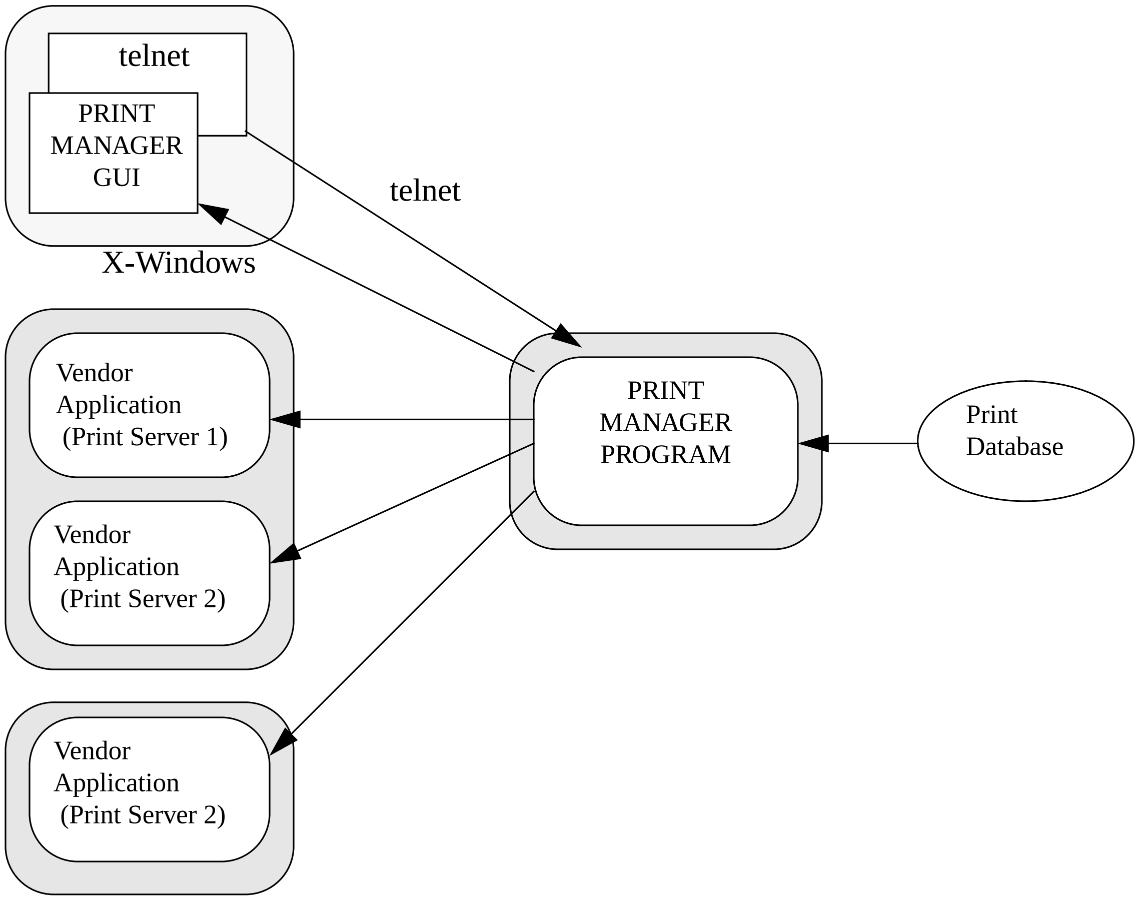

telnet
PRINT MANAGER GUI
telnet
X-Windows
Vendor Application (Print Server 1)
Vendor Application (Print Server 2)
Vendor Application (Print Server 2)
PRINT MANAGER PROGRAM
Print Database

FIGURE 7. Print Manager Program Connections

Figure 7 shows a TELNET session with a CTN running the print manager. The CTN has access to one print database. Also shown are possible connections between the print manager and several print servers.

5.1 Running the Print Manager Program

For the demonstration, vendors can use TELNET to connect to the CTN and log in using a predefined UNIX account. Each vendor is given their own UNIX login. After successfully logging in on a CTN, a script file is executed. This script displays options available to the vendor, and one of the option is to run *pmgr_motif*.

5.1.1 Command Line Arguments For The Print Manager Program

The print manager is invoked from the command line with a number of switches one one mandatory argument:

pmgr_motif [switches] printerGroup

-a appTitle	Use appTitle as the AE Title of the print manager instead of the default PRINT_MANAGER).
-D database	Set the print database name (default: print.db)
-f control	Set the control database name (default: CTNControl)
-F file	Set the icon file pathname (default: icon.file)
-h	Print help page
-I index	Set the icon index pathname (default: icon.index)
-s	Place the print manager in silent mode
-S	Set Basic Image Box attributes as user creates each BIB rather than waiting until the end.
-v	Place all facilities in verbose mode.
-x fac	Place one facility in verbose mode (fac = DUL, SRV, DCM, TBL)
printerGroup	The group name which identifies the set of printers that can be targets for this invocation of the print manager.

There is only one required argument for running *pmgr_motif*. This is the name of the group for all prospective target print servers. At a demonstration, the group name (server group) is typically a company name and will be used to identify all of the print servers run by that company.

All optional switches must appear before the server group, but their order is arbitrary.

The -a switch assigns an Application Entity title to the print manager. Default value is PRINT_MANAGER.

The -D switch identifies the name of the database file contains patient and study information. This database is not built with Sybase but uses the database software that was developed for 1993. The document “Users’s Guide for Print Utilities” tells how to build this database.

The -f switch allows the print manager to use a different control database. The default is CTNControl.

The -I switch defines an icon index which is needed to present the 64 x 64 icons. That file is described in “User’s Guide for Print Utilities.”

Normally the print manager will print a number of status messages and data sets as it communicates with a print server. If you use the -s switch, this information will be suppressed.

The -S switch controls the process for setting attributes for each basic image box. The default is to set all attributes for all BIBs when the user finally selects the “Print” button. The -S switch tells the print manager to set the attributes for each BIB as the user selects each image.

The -v switch places a number of facilities in verbose mode (DCM, DUL, SRV, TBL)

The -x switch can place one facility in verbose mode.

5.1.2 Configuring Print Manager to Connect with External Nodes

The print manager uses two tables in the Control Database to connect to external nodes. The GroupNames table is used to map a set of print servers into one group name. In order to define the applications in the print group, use cfg_ctn_tables and select Group from the pulldown menu General. (For example, we use the group name ERL_PRINTERS and define several application entities under that group).

The print manager uses the ApplicationEntity control table to map the selected print server (Application Entity Title) into a host name and a TCP/IP port number.

The print manager uses two different database libraries. The TBL facility is used to handle the configuration data while older libraries are used to handle the actual data. The database tables for handling the print data and icons are of local design and do not use the relational database implementations.

5.2 The Print Manager User Interface

The Print Manager provides several displays for different tasks. These are presented to the user in order to create a print session and print images.

- The Print Manager Window displays the list of print servers the user can select. When the user selects a print server, an Association is requested. The print server displays a window which allows the user to request some characteristics of the print server and displays a window for creating a Basic Film Session.
- The Basic Film Session Window allows the user to select parameters for a film session and create a film session.
- The Basic Film Box Window allows the user to select the parameters for a film box and creates the film box.
- The Study Window displays all the studies existing in the database. The Icon Selection Window displays the currently selected study’s images, each in an iconified representation. The Film Window is a simple representation of the film that will be printed. From this window, the vendor can place a selected image on the desired location on the film, the selected image being the icon selected in the Icon Selection Window. The Image Attribute Window provides the interface that allows the vendor to choose options specific to an image selected in the Film Window. Following are some additional tips for selecting images to be printed.
- Select a study from the study list.
- Select an image form the icon window, and that image will be transferred to the film window. If you want to place the image in a specific location of the film, select that location in the film window before selecting the image.

- Select multiple image using the middle button of the mouse. (You can only select a consecutive set of image) Select the first image with the middle button, and then select the last image, again with the middle button.

5.3 start_print Script

The *start_print* script is used to provide the parameters needed to start the Print Manager. It assumes that a root directory exists with configuration and database files and starts *pmgr_motif* with the proper command-line arguments. The single argument to start_print is a group name. Look in the script for the legal group names (and modify the script if necessary for your purposes).

6.0 Print Server

The CTN contains a print server program, *print_server*, which is designed to demonstrate the print management service provided by the performing SCU (SCP). The print server program provides print management service for the Basic Print Management Meta SOP classes. The print server program does not offer support for printing images on a hard copy device. Instead, it is provided with a companion program, *print_server_display*, that emulates a hard copy device by displaying the images on an X-Window display terminal.

This is supported in the Unix versions of the software but not the Windows version.

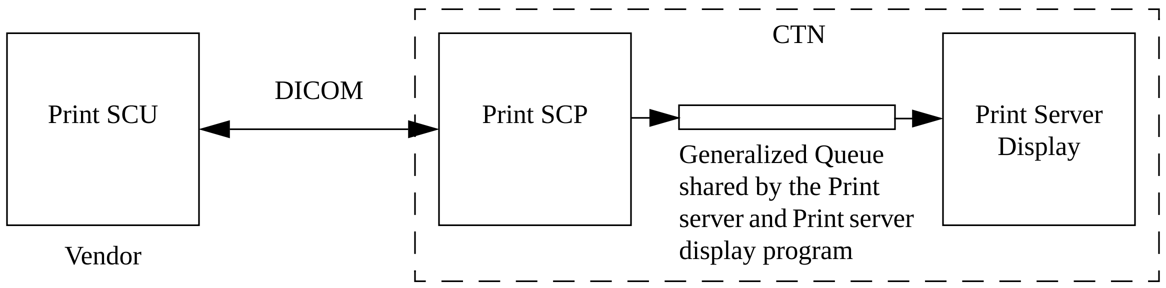

CTN
Print SCU
DICOM
Print SCP
Print Server Display
Generalized Queue shared by the Print server and Print server display program
Vendor

FIGURE 8. Interaction between the Print Server and Print Display

6.1 Print Server Control Tables

The print server program uses the ApplicationEntity and SecurityMatrix tables described earlier in this document in the section describing the image server.

6.2 Print Server Configuration

In a future release, the print server will be configurable to use a different queue ID depending on which application initiates the association. In this release, the queue ID is a command line parameter. We suggest you use the value 1.

6.3 Invoking the Print Server and Print Server Display

The Print Server and the Print Server Display programs communicate with each other via a shared queue. Incoming requests from the print client are enqueued in the shared queue by the print server. The print server display program dequeues these requests from the shared queue and proceeds to display the type of request made. The queue shared by these two programs is identified by a unique queue number which has to be specified as a command line argument by both the programs at invocation time. The print server uses two environment variables to access files for generating unique identifiers and for accessing the display queue. Set the environment variable QUEUE_DIRECTORY to point to a directory where the GQ routines can create a file for queue access. The environment variable UIDFILE should give the path name of a UID file as described in User's Guide to the UIDFacility. An example print uid file is found in *uids/print_server.uid*.

A summary page of the print server program's command line arguments can be displayed by running the print server program (print_server) from the UNIX command line without any arguments. The print server program can be invoked as shown below:

print_server [optional switches] port_number

The print server program requires one mandatory argument, *port_number*, which is used by the *print_server* to listen to incoming print requests. The print server supports the following optional switches:

-d <facility>	Puts the desired facility in verbose mode
-f	Ignores trivial errors in the association request
-n <node>	Uses node instead of the hostname
-r	To send the attribute data set to the requester via the response messages
-s	Make the server iterative i.e. service requests one at a time
-t	Sets the trace flag
-x <id>	Uses the GQ facility to maintain a queue used by print_server_display
-v	Puts the DUL and SRV facilities in verbose mode

The print server display program can be invoked as shown below:

print_server_display [optional switches] GQ_ID

The print_server_display program requires one mandatory argument which is the GQ ID used by the print_server program. The print_server_display also supports the following optional switches:

-w width	Sets the width of the display window
-h height	Sets the height of the display window
-v	Sets the verbose mode

The default size of the display window is the size of the screen. This size could be manipulated using the -w and -h switches of the print_server_display program. The print server display program needs the same queue directory that is specified for the print server program. The print server display program does not care about UID files.

6.4 Summary of Steps for Using the Print Server

This section provides a summary of the steps that are necessary to configure and run the print server program.

1. Create the ApplicationEntity and the SecurityMatrix tables for later use by the print server program.
2. Create an entry in the ApplicationEntity table for the print server. We suggest you use the application title MIR_PRINT_SCP.
3. Create enries in the ApplicationEntity table for your print clients.

4. Create one entry in the SecurityMatrix table for each print client that will request an Association with the MIR print server.
5. Make certain you have the appropriate environment variables set for the GQ and UID facilities.
6. Invoke the print_server program with any optional switches and the mandatory port_number argument. If the print_server_display program is also going to be used, then the GQ facility (-x <GQ ID>) switch must be used.
7. The print_server_display program could be used if the print_server's -x switch is used. The GQ ID used by the *print_server* and *print_server_display* must be the same.

7.0 Utilities for CTN Demonstration Applications

CTN demonstration applications may use one or more utility programs for configuration or data loading. This section describes common applications that are used to initialize data for demonstration applications.

These applications are supported in the Unix versions of the software but not the Windows version.

7.1 Populating an Image Database with *fillImageDB*

fillImageDB is a utility program that is provided to place data in an image database for a set of existing files. *fillImageDB* does no sorting and little verification. It assumes the data is already stored in the files in "CTN" format (not Part 10 compliant). Invoke *fillImageDB* as follows:

```
fillImageDB [-o owner] database file [file...]
```

We use the -o switch to set the owner attribute at the Patient, Study, Series and Image levels in our database. We use this for sorting, but it is not essential. *database* is the database name (not the AE title of an application). You can specify one or more files to be placed in the database (e.g., we typically use .../images/vendor/*/*). You will want to specify a full pathname for image files so that the images can be opened by a server. Using a relative pathname would place restrictions on how you start the CTN demonstration applications.

The one warning is that the underlying IDB functions may try to delete image files that have already been loaded into the database. This means that if you get part way through loading the images and fail, you should clear the database first before retrying. Otherwise, when you rerun your script to load all of the images, the files which have already been loaded will be deleted.

We also have a script which assumes the files are stored as follows:

```
.../vendor/study1/images
.../vendor/study2/images
```

The *fillImageDBScript* will go through a set of directories and repeatedly invoke *fillImageDB* with the -o switch to load the database. The same warning about failure and repeat loads applies to this script as well.

We use this application and script for populating the image database for the archive server and for the modality emulator.