Programmer's Guide to the IDB Facility

A Facility for Manipulating the DICOM Hierarchical Query Model

David E. Beecher

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

Version 2.10.0
August 3, 1998

1 Introduction

The IDB routines provide a structured access mechanism for building and maintaining a DICOM hierarchical data model.. This library is not database dependent, does rely quite heavily on the TBL facility. This library include routines to open and close individual databases (IDB_Open and IDB_Close), as well as image insertion (IDB_InsertImage and IDB_InsertImageInstance), deletion (IDB_Delete), and selection (IDB_Select).

Recall that the DICOM data model includes Patient, Study, Series, Image levels. The insertion routine is simplified by not having a separate routine at each level, but rather one routine that handles all levels. Furthermore, the concept of an image instance has been introduced which allows users of this facility to store multiple “instances” or copies of the same image in the database. This is useful to differentiate between different storage mechanisms, high-speed vs. low-speed, etc.. This library was designed primarily for the support of the DICOM Image Server.

2 Data Structures

idb.h is the primary include file for applications wishing to use the facility. There are several data structures defined which are of use to the developer. The first group of structures presented were designed for the selection routine. Notice that in each structure several fields are present that are not part of the DICOM data model. Several fields have been added to facilitate the maintenance of this database, like InsertDate and InsertTime, as well as parent node pointers to correctly maintain the hierarchical connections.

```
typedef struct _IDB_PatentQuery {
    char
        PatNam[IDB_PN_QLENGTH+1],
        PatID[IDB_LO_QLENGTH+1],
        PatBirDat[IDB_DA_QLENGTH+1],
        PatBirTim[IDB_TM_QLENGTH+1],
        PatSex[IDB_CS_QLENGTH+1];
    long
        NumPatRelStu,
        NumPatRelSer,
        NumPatRelIma;
    char
        InsertDate[IDB_DA_QLENGTH+1],
        InsertTime[IDB_TM_QLENGTH+1],
        Owner[IDB_OWNER_QLENGTH+1],
        GroupName[IDB_GROUP_QLENGTH+1],
        Priv[IDB_PRIV_QLENGTH+1];
} IDB_PatientQuery;
```

```
typedef struct _IDB_StudyQuery {
    char
        StuDat[IDB_DA_QLENGTH+1],
        StuTim[IDB_TM_QLENGTH+1],
        AccNum[IDB_SH_QLENGTH+1],
        StuID[IDB_SH_QLENGTH+1],
        StuInsUID[IDB_UI_QLENGTH+1],
        RefPhyNam[IDB_PN_QLENGTH+1],
        StuDes[IDB_LO_QLENGTH+1],
        PatAge[IDB_AS_QLENGTH+1],
        PatSiz[IDB_DS_QLENGTH+1],
        PatWei[IDB_DS_QLENGTH+1];
    int
        NumStuRelSer,
        NumStuRelIma;
    char
        InsertDate[IDB_DA_QLENGTH+1],
        InsertTime[IDB_TM_QLENGTH+1],
        Owner[IDB_OWNER_QLENGTH+1],
        GroupName[IDB_GROUP_QLENGTH+1],
        Priv[IDB_PRIV_QLENGTH+1];
    char
        __PatParent__[DICOM_UI_LENGTH+1];
} IDB_StudyQuery;



typedef struct _IDB_SeriesQuery {
    char
        Mod[IDB_CS_QLENGTH+1],
        SerNum[IDB_IS_QLENGTH+1],
        SerInsUID[IDB_UI_QLENGTH+1],
        ProNam[IDB_LO_QLENGTH+1],
        SerDes[IDB_LO_QLENGTH+1],
        BodParExa[IDB_CS_QLENGTH+1],
        StuDes[IDB_LO_QLENGTH+1];
            int
        NumSerRelIma;
    char
        InsertDate[IDB_DA_QLENGTH+1],
        InsertTime[IDB_TM_QLENGTH+1],
        Owner[IDB_OWNER_QLENGTH+1],
        GroupName[IDB_GROUP_QLENGTH+1],
        Priv[IDB_PRIV_QLENGTH+1];
    char
__StuParent__[DICOM_UI_LENGTH+1];
} IDB_SeriesQuery;
```

```
typedef struct _IDB_ImageQuery {
    char
        ImaNum[IDB_IS_QLENGTH+1],
        SOPInsUID[IDB_UI_QLENGTH+1],
        SOPClaUID[IDB_UI_QLENGTH+1],
        PhoInt[IDB_CS_QLENGTH+1];
    int
        SamPerPix,
        Row,
        Col,
        BitAll,
        BitSto,
        PixRep;
    char
        InsertDate[IDB_DA_QLENGTH+1],
        InsertTime[IDB_TM_QLENGTH+1],
        Owner[IDB_OWNER_QLENGTH+1],
        GroupName[IDB_GROUP_QLENGTH+1],
        Priv[IDB_PRIV_QLENGTH+1];
    char
        __SerParent__[DICOM_UI_LENGTH+1];
    LST_HEAD
        *ImageUIDList,
        *InstanceList;
} IDB_ImageQuery;
```

```
typedef struct _IDB_Query {
   IDB_PatientQuery patient;
   IDB_StudyQuery study;
   IDB_SeriesQuery series;
   IDB_ImageQuery image;
   long
         PatientQFlag,
         StudyQFlag,
         SeriesQFlag,
         ImageQFlag;
         PatientNullFlag,
         StudyNullFlag,
         SeriesNullFlag,
         ImageNullFlag;
} IDB_Query;
```

The following bit-flags are defined for the IDB_Query structure to signal which fields should be examined for retrieval:

```
/*
 * Query Flags for IDB_Select--Patient Level
 */
#define QF_PAT_PatNam           0x00000001
#define QF_PAT_PatID            0x00000002
#define QF_PAT_PatBirDat        0x00000004
#define QF_PAT_PatBirTim        0x00000008
#define QF_PAT_PatSex           0x00000010
#define QF_PAT_NumPatRelStu     0x00000020
#define QF_PAT_NumPatRelSer     0x00000040
#define QF_PAT_NumPatRelIma     0x00000080
#define QF_PAT_InsertDate       0x00000100
#define QF_PAT_InsertTime       0x00000200
#define QF_PAT_Owner            0x00000400
#define QF_PAT_GroupName        0x00000800
#define QF_PAT_Priv             0x00001000
/*
 * Query Flags for IDB_Select--Study Level
 */
#define QF_STU_StuDat           0x00000001
#define QF_STU_StuTim           0x00000002
#define QF_STU_AccNum           0x00000004
#define QF_STU_StuID            0x00000008
#define QF_STU_StuInsUID        0x00000010
#define QF_STU_RefPhyNam        0x00000020
#define QF_STU_StuDes           0x00000040
#define QF_STU_PatAge           0x00000080
#define QF_STU_PatSiz           0x00000100
#define QF_STU_PatWei           0x00000200
#define QF_STU_NumStuRelSer     0x00000300
#define QF_STU_NumStuRelIma     0x00000400
#define QF_STU_InsertDate       0x00000800
#define QF_STU_InsertTime       0x00001000
#define QF_STU_Owner            0x00002000
#define QF_STU_GroupName        0x00004000
#define QF_STU_Priv             0x00008000
/*
 * Query Flags for IDB_Select--Series Level
 */
#define QF_SER_Mod              0x00000001
#define QF_SER_SerNum           0x00000002
#define QF_SER_SerInsUID        0x00000004
#define QF_SER_ProNam           0x00000008
#define QF_SER_SerDes           0x00000010
#define QF_SER_BodParExa        0x00000020
#define QF_SER_NumSerRelIma     0x00000040
#define QF_SER_InsertDate       0x00000080
#define QF_SER_InsertTime       0x00000100
#define QF_SER_Owner            0x00000200
#define QF_SER_GroupName        0x00000400
#define QF_SER_Priv             0x00000800
/*
 * Query Flags for IDB_Select--Image Level
 */
```

```
#define QF_IMA_ImaNum           0x00000001
#define QF_IMA_SOPInsUID        0x00000002
#define QF_IMA_SOPClaUID        0x00000004
#define QF_IMA_SamPerPix        0x00000008
#define QF_IMA_PhoInt           0x00000010
#define QF_IMA_Row              0x00000020
#define QF_IMA_Col              0x00000040
#define QF_IMA_BitAll           0x00000080
#define QF_IMA_BitSto           0x00000100
#define QF_IMA_PixRep           0x00000200
#define QF_IMA_InsertDate       0x00000400
#define QF_IMA_InsertTime       0x00000800
#define QF_IMA_Owner            0x00001000
#define QF_IMA_GroupName        0x00002000
#define QF_IMA_Priv             0x00004000
#define QF_IMA_SOPInsUIDList    0x00008000
```

Insertion is handled a little differently, due to the fact that not all the fields can be inserted by the user. Different structures were designed to accomodate this and are described below. For example, the user, when inserting and new image, is not allowed to set the InsertDate or InsertTime field, the insertion routines handle that automatically.

```
typedef struct _IDB_PatientNode {
    char
        PatNam[DICOM_PN_LENGTH + 1],
        PatID[DICOM_LO_LENGTH + 1],
        PatBirDat[DICOM_DA_LENGTH + 1],
        PatBirTim[DICOM_TM_LENGTH + 1],
        PatSex[DICOM_CS_LENGTH + 1];
    char
        Owner[IDB_OWNER_LENGTH + 1],
        GroupName[IDB_GROUP_LENGTH + 1],
        Priv[IDB_PRIV_LENGTH + 1];
}   IDB_PatientNode;

typedef struct _IDB_StudyNode {
    char
        StuDat[DICOM_DA_LENGTH + 1],
        StuTim[DICOM_TM_LENGTH + 1],
        AccNum[DICOM_SH_LENGTH + 1],
        StuID[DICOM_SH_LENGTH + 1],
        StuInsUID[DICOM_UI_LENGTH + 1],
        RefPhyNam[DICOM_PN_LENGTH + 1],
        StuDes[DICOM_LO_LENGTH + 1],
        PatAge[DICOM_AS_LENGTH + 1],
        PatSiz[DICOM_DS_LENGTH + 1],
        PatWei[DICOM_DS_LENGTH + 1];
    char
        Owner[IDB_OWNER_LENGTH + 1],
        GroupName[IDB_GROUP_LENGTH + 1],
        Priv[IDB_PRIV_LENGTH + 1];
}   IDB_StudyNode;
```

```
typedef struct _IDB_SeriesNode {
    char
        Mod[DICOM_CS_LENGTH + 1],
        SerNum[DICOM_IS_LENGTH + 1],
        SerInsUID[DICOM_UI_LENGTH + 1],
        ProNam[DICOM_LO_LENGTH + 1],
        SerDes[DICOM_LO_LENGTH + 1],
        BodParExa[DICOM_CS_LENGTH + 1];
    char
        Owner[IDB_OWNER_LENGTH + 1],
        GroupName[IDB_GROUP_LENGTH + 1],
        Priv[IDB_PRIV_LENGTH + 1];
}   IDB_SeriesNode;

typedef struct _IDB_ImageNode {
    char
        ImaNum[DICOM_IS_LENGTH + 1],
        SOPInsUID[DICOM_UI_LENGTH + 1],
        SOPClaUID[DICOM_UI_LENGTH + 1],
        PhoInt[DICOM_CS_LENGTH + 1];
    int
        SamPerPix,
        Row,
        Col,
        BitAll,
        BitSto,
        PixRep;
    char
        Owner[IDB_OWNER_LENGTH + 1],
        GroupName[IDB_GROUP_LENGTH + 1],
        Priv[IDB_PRIV_LENGTH + 1];
    char
        RespondingTitle[17],
        Medium[33],
        Path[256],
        Transfer[65];
    int
        Size;
}   IDB_ImageNode;

typedef struct _IDB_Insertion {
    IDB_PatientNode patient;
    IDB_StudyNode study;
    IDB_SeriesNode series;
    IDB_ImageNode image;
} IDB_Insertion;
```

These data structures are referenced in the routine descriptions that follow.

3 Include Files

Any applications needing to use this facility should include the following files:

```
#include “idb.h”
```

4 Return Values

The following returns are defined from the IDB routines:

IDB_NORMAL	Operation completed successfully
IDB_UMIMPLEMENTED	The operation attempted is currently unimplemented
IDB_ALREADYOPENED	The specified database is already opened
IDB_BADDBTABPAIR	For each database opened, a certain number of tables within that database must exist and open successfully
IDB_NOMEMORY	Unable to dynamically allocate needed memory.
IDB_CLOSERROR	An error occurred attempting to close a database
IDB_BADHANDLE	The handle passed is invalid
IDB_BADLEVEL	The DICOM level specified is invalid
IDB_NULLUID	A null UID was passed
IDB_BADPATUID	An invalid Patient UID was passed
IDB_BADSTUUID	An invalid Study UID was passed
IDB_BADSERUID	An invalid Series UID was passed
IDB_BADIMAUID	An invalid Image UID was passed
IDB_BADLISTENQ	An attempt to add a node to an internal list failed
IDB_NOINSERTDATA	No data was provided to insert
IDB_BADLEVELSEQ	A bad BEGIN/END level sequence was passed
IDB_NOMATCHES	No database matches were found for the query
IDB_EARLYEXIT	The user’s callback routine returned something other than IDB_NORMAL which caused the select to quit early
IDB_DUPINSTANCE	Attempt to insert a duplicate instance in the database

5 IDB Routines

Detailed descriptions of the IDB functions are included in this section.

IDB_Close

Name

IDB_Close -this routine closes a previously opened database

Synopsis

CONDITION IDB_Close(char *databaseName, IDB_HANDLE **handle)

databaseName	The name of the database to open.
handle	contains the database handle

Description

This routine attempts to find the handle in it's internal table of open database descriptors and closes all the tables associated with that descriptor.

Notes

None

Return Values

```
IDB_NORMAL
IDB_CLOSERROR
```

IDB_Delete

Name

IDB_Delete -this routine deletes node(s) in the hierarchy starting at the node of the selected UID

Synopsis

CONDITION IDB_Delete(IDB_HANDLE **handle, long level, char *uid)

handle	the database identifier.
level	The level in the hierarchy specifying where the next parameter, uid, will be found. level must be one of the pre-defined constants, `IDB_PATIENT_LEVEL`, `IDB_STUDY_LEVEL`, `IDB_SERIES_LEVEL`, or `IDB_IMAGE_LEVEL`.
uid	specifies the uid of the node at which to begin the deletion.

Description

IDB_Delete creates lists of all the uids to be deleted and then simply issues the appropriate TBL_Delete calls to perform that task. It also updates counts in the un-deleted nodes where appropriate.

Notes

None.

Return Values

```
IDB_NORMAL
IDB_BADHANDLE
IDB_BADLEVEL
IDB_NULLUID
IDB_BADPATID
IDB_BADSTUUID
IDB_BADSERUID
IDB_BADIMAUID
IDB_NOMEMORY
IDB_BADLISTENQ
```

Name

IDB_InsertImage -this routine inserts records into the database

Synopsis

CONDITION IDB_InsertImage(IDB_HANDLE **handle, IDB_Insertion *pssi)

handle	the database identifier
pssi	the structure that contains the new record to be inserted into the database

Description

The insertion algorithm first check to determine if any of the uids passed in pssi are contianed in the database. If so, then these levels need not be replaced...simply updated with new counts for the number of descendants. If multiple records with that UID exist, a database integrity problem exists. This routine generates the appropriate error and the insertion is aborted.

Notes

None.

Return Values

```
IDB_NORMAL
IDB_NOINSERTDATA
IDB_BADPATUID
IDB_BADSTUUID
IDB_BADSERUID
IDB_BADIMAUID
IDB_DUPINSTANCE
```

IDB_InsertImageInstance

Name

IDB_InsertImageInstance -this routine inserts an image instance record into the database

Synopsis

CONDITION IDB_InsertImageInstance(IDB_HANDLE **handle, char *imageuid, IDB_InstanceListElement *iie)

handle	the database identifier
imageuid	the image uid for which the instances will be inserted.
iie	the list of instances to be inserted.

Description

The image UID (imageuid) passed must exist. The routine then inserts the instance(s) into the image instance table.

Notes

None.

Return Values

```
IDB_NORMAL
IDB_BADHANDLE
IDB_BADIMAUID
```

Name

IDB_Open -this routine attempts to open for access the database pointed to by the input string databaseName.

Synopsis

CONDITION IDB_Open(char *databaseName, IDB_HANDLE **handle)

databaseName	the name of the database to open.
handle	will contain the newly opened database handle upon success

Description

IDB_Open uses the TBL facility extensively to determine if the needed tables can be opened and accessed. If so, this routine allocates a context which contains pointers to the tables just opened and saves this context in a linked list maintained by IDB_Open and IDB_Close.

Notes

None.

Return Values

```
IDB_NORMAL
IDB_ALREADYOPENED
IDB_BADDBTABPAIR
IDB_NOMEMORY
```

Name

IDB_Select -this routine selects records from the database and uses the DICOM matching specifications for retrieval

Synopsis

```
CONDITION IDB_Select( IDB_HANDLE **handle, IDB_QUERY_MODEL model
                      long begin_level, long end_level, IDB_QUERY *pssi, long *count,
                      CONDITION (*callback()), void *ctx )
```

handle	the database identifier.
model	The DICOM query model to be used for the query. One of an enumerated set including PATIENT_ROOT, STUDY_ROOT, and PATIENTSTUDY_ONLY.
begin_level *end_level*	the levels in the hierarchy specifying where the search for records will begin and end. begin_level and end_level must be one the pre-defined constants: `IDB_PATIENT_LEVEL`, `IDB_STUDY_LEVEL`, `IDB_SERIES_LEVEL`, or `IDB_IMAGE_LEVEL` (see Notes below).
count	this parameter will contain the count of the number of records matched upon return.
callback	the callback function invoked when a matching record is found. It is invoked as described below.
ctx	ancillary data passed through to the callback function and untouched by this routine.

Description

As each record is retrieved from the database, the fields requested by the user (contained in pssi), are filled with the information retrieved from the database and a pointer to the list is passed to the callback routine designated by the input parameter callback. The callback routine is invoked as follows:

```
callback( IDB_Query *pssi, long count, void  *ctx)
```

Count contains the number of records retrieved to this point. *Ctx* contains any additional information the user originally passed to the select function. If *callback* returns any value other than `IDB_NORMAL`, it is assumed that this function should terminate (i.e. cancel the current db operation), and return an abnormal termination message (`IDB_EARLYEXIT`) to the routine which originally invoked the select.

Notes

The addition of *model* to this routine allows for a more efficient implementation of the STUDY_ROOT retrieval that was possible before. The user should remember that even if the STUDY_ROOT model is chosen, patient information is only returned if the begin_level has a value of IDB_PATIENT_LEVEL. Even though (logically) the patient and study levels are collapsed in the STUDY_ROOT model, internally they are still stored separately.

This routine contains the use of a "go to" to implement the structure construct known as a multi-level break statement. 'c' has a single level break statement in the language but no facility to implement a multi-level break. This algorithm could well have been implemented without using the actual "go to", but the resulting code would have been more difficult to read and maintain in my opinion. I am not fond of using "go to's", and rarely ever do, but I do find that every 100 thousand lines or so that the need arises...

Return Values

IDB_NORMAL
IDB_BADHANDLE
IDB_BADLEVEL
IDB_BADLEVELSEQ
IDB_NOMATCHES
IDB_EARLYEXIT
IDB_NOMEMORY