Programmer's Guide to the DCM Facility

A Facility for Manipulating DICOM Information Objects

Stephen M. Moore

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

This document describes a general software facility for manipulating DICOM V3 Information Objects.

---

1 Introduction

The DCM facility is a set of routines used to manipulate DICOM Information Objects defined by the DICOM V3.0 (Draft) Standard. These routines are based on Parts 5 and 6 of the Standard. They allow the user to create, parse, update, import and export Information Objects. These routines do not enforce any rules which are defined in Part 3 of the Standard.

1.1 Data Elements

The V3 standard defines a number of data elements which are identified by tag (concatenation of group and element number), length and data value. Each data element is defined by the 3-tuple (tag, length, value). Tag is defined as a 32 bit unsigned integer. length is a 32 bit unsigned integer and is the length in bytes of the value to follow. Each data element has an associated data representation which is defined in the dictionary in the V3 standard. Part 5 defines the list of possible representations.

Several representation are included below as an example.

AS Age String

DA Data

PN Person Name

SS Signed Short

The DCM facility provides means for informing the application of the representation of the data. The DCM facility will also make certain that an application does not attempt to create objects with illegal data elements. That is, if the data element appears in the DCM dictionary, the application must use the proper value representation when creating objects or ask to use the representation found in the dictionary. The facility does not examine ASCII data elements to verify that numeric and text fields are formatted properly.

The V3 standard defines whether data elements contain a single value or multiple values. This is referred to as value multiplicity. This facility provides no direct mechanisms for supporting data elements with multiple values and leaves it to the application to interpret data elements that assume multiple values.

1.2 Data Objects

Data elements are combined to form objects. Data elements are stored in sequential order by tag value. This facility defines three types of objects:

COMMAND	An object consisting only of data elements in the Command group (0000). Part 7 of the V3 standard lists the commands that are defined.
DATA SET	An object consisting of groups excluding the Command group. An example would be an image or an identifier for a query.
UNKNOWN	An object which is empty or contains both command and non-command elements.

Objects are maintained by the facility in an internal format that is hidden from the application. For the rest of the Introduction, a reference to object will mean the internal representation which is maintained by the DCM facility during the execution of an application. We will not use object to refer to a DICOM V3 stream that might be stored in a file or in a buffer in the application's data space.

This facility provides several mechanisms for creating and modifying objects. The user may create an object by:

- Opening a file that contains an DICOM V3 data stream.
- Importing a data stream from the application's virtual memory that contains DICOM V3 data.
- Creating an object explicitly and adding a number of data elements to the object.

Once an object has been created, it may be manipulated with the following types of functions:

- Add a new data element to the object.
- Remove a data element from the object.
- Modify the value of existing elements in the object.
- Extract the size of the object when represented in stream format.
- Extract the size of a single data element.
- Extract the data from a single data element.
- Export the object into the V3 stream format (suitable for writing to disk or over a network).
- Print a human-readable version of the object to the standard output.

Please refer to the individual function pages for a complete list and description of the available functions.

Part 3 of the V3 Standard describes Information Objects and defines which elements are required (mandatory) or optional. This facility provides no means for verifying that an object meets the V3 standard by containing the required data elements. That is, the DCM facility does not examine objects to make certain they contain mandatory data elements. This facility will maintain data

elements in proper numerical order, but the application must determine that an object has the required data elements.

The group length data element (element 0000 in any group) has been either retired or made optial. This implementation accepts Information Objects with group length elements, but does not maintain these elements. The export routines do not write group length elements even if they are in the original object.

1.3 Miscellaneous Issues

The V3 standard defines a number of transfer syntaxes which are used when data is placed in the stream format. This applies to files that store DICOM V3 data and to stream buffers that are imported and exported by this facility. The DCM facility provides options to allow the user to specify the byte order of the data. That is, an application can work with stream data that is in little-endian or big-endian format. The little-endian format corresponds to the "DICOM Implicit VR Little Endian Transfer Syntax" defined by Part 5 of the Standard. The big-endian format is used for old V2 data files for manufacturers that stored data in big-endian format. The other transfer syntaxes defined in Part 5 and the file format defined in Part 10 are not yet implemented. Whether the data is represented in little-endian or big-endian format, the items in a data element are always transmitted in the following order:

- Group
- Element
- Length
- Value

When an application wishes to add a single data element to an object or to retrieve the data from a single element, this facility uses the byte order that is native to the machine. In this way, the facility tries to minimize the work an application has to do to interpret the data values.

Data elements that are not present in the data dictionary present special problems. This facility provides no means for extending the data dictionary. If a file with DICOM data contains elements that are not in the data dictionary, the DCM facility will not be able to assist applications with value representation and byte order. It will be up to the application to know how to interpret such data. When individual data elements are added to an existing object, the application may specify the data representation for data elements not in the dictionary. However, this specification is only maintained as long as the object is maintained by the package. Once the data is exported to a stream or to a file, the DCM facility will not know how to interpret the data. With this implementation of the DCM facility, the safest practice for data elements not in the data dictionary is to tell the package they are of unknown representation (DCM_UNKNOWN). The package will then pass the data without trying to interpret or modify the order of the bytes. (This will change when Part 10 is implemented).

When the facility opens a file containing DICOM data for the purpose of creating an object, the file is opened for read-only access. If the application modifies the object, those modifica-

tions are local to the current process and are not written back to the file. When a file is opened, most of the data elements are read into virtual memory for fast recall by applications. Some data elements remain on disk. Specifically, this facility does not read the pixel data immediately. This is done for performance reasons. The facility maintains a pointer to the pixel data in the file and will retrieve the pixels when asked. In this way, an application does not have to pay a performance penalty when opening a file containing a large image. The result of this is that the DICOM file may remain open after the call to DCM_OpenFile until you explicitly close that object (DCM_CloseObject). This means that applications will not want to have a large number of objects which were created from files because the processes are likely to run out of file descriptors. System administrators have more information on available file descriptors.

DICOM V3 defines a number of value representations. The current version of the software does not implement the SQ representation. This will be corrected shortly.

2 Data Structures

The include file with the DCM facility defines two data structures of importance to the user of this package. A DCM_OBJECT is a handle used by the caller to hold an object. This handle is private to the DCM facility and cannot be examined by the caller. Users should declare their variables as a pointer to a DCM_OBJECT (DCM_OBJECT *obj) and pass the variables by reference (&obj). All functions in this facility expect the user to pass the reference to an object with this mechanism.

2.1 DCM_ELEMENT

The include file also defines a DCM_ELEMENT as a structure:

```
typedef struct {
    DCM_TAG               tag;
    DCM_VALUEREPRESENTATION    representation;
    char                  description[48];
    unsigned long            multiplicity;
    unsigned long            length;
    union {
        char *string;
        char **stringArray;
      short *ss;
      long *sl;
      unsigned short *us;
      unsigned long *ul;
      unsigned char *ob;
      unsigned short *ow;
      LST_HEAD *sq;
      DCM_TAG *at;
    } d;
} DCM_ELEMENT;
```

This is the structure used to describe individual data elements and to pass them into and out of the package. The caller specifies a data element by filling in the tag field. The include file defines macros for data elements found in the data dictionary. It is suggested that programmers use the macros instead of hard-coding hexadecimal values. length gives the length of the data element. length may be provided by the caller when adding a data element to an object or may be provided by the facility when the caller requests the length of a data element. Please refer to the function descriptions for the use of this field.

The representation is an enumerated type which is defined in the include file for the DCM facility. Valid representations are:

DCM_AE	DCM_AS	DCM_AT	DCM_CS	DCM_DA
DCM_DD	DCM_DS	DCM_DT	DCM_FD	DCM_FL
DCM_IS	DCM_LO	DCM_LT	DCM_OB	DCM_OW
DCM_PN	DCM_SS	DCM_SH	DCM_SL	DCM_SQ
DCM_ST	DCM_TM	DCM_UI	DCM_UL	DCM_US
DCM_CTX	DCM_RET	DCM_UNKNOWN		

Most of the representations listed above are found in Part 5 of the Standard. DCM_UNKNOWN is used as a tag for unknown representations. DCM_RET is used for retired attributes. DCM_CTX is used for context sensitive attributes. Depending on the function, this field may be filled in by the facility or by the application. Please refer to the individual functions for the use of this field.

The englishDescription field is a short text field that contains a human-readable description of the data element. This is most helpful for developers and is also used when the facility is asked to print the contents of an object. This field is normally filled in by the DCM facility. There are some functions that allow the caller to fill in this field.

The d field provides a pointer to the actual data value. This field is always written by the caller and is never written by the DCM facility. This means that the caller is always responsible for storage when adding data elements or when extracting data elements from an object. When the caller asks the facility to retrieve the data value for a data element, the DCM facility requires that the user set aside storage for that data value. This minimizes confusion over the owner of memory that might be allocated. When the caller adds an element to an object, the facility will make a copy of the data value, so the caller need not maintain the data once the data element has been added.

The d field is a union of pointers of different types. The DCM facility allows the user to reference data using the appropriate pointer for the data type. Since a number of data types are "string types", we collapse that definition into a single pointer, string. When a caller allocates data for data, they should take care to maintain the data on proper address boundaries. For example, if a

function passes the address of data which is an unsigned long (4 bytes), the data should be aligned on a word boundary and not on an odd boundary.

3 Include Files

Any source code that uses the DCM routines, structure definitions or constants should include the following files in the order given:

```
#include “dicom.h”
#include “lst.h”
#include “dicom_objects.h”
```

4 Return Values

DCM_BADELEMENTINGROUP	DCM software detected an internal error where an element was placed in the wrong group.
DCM_CALLBACKABORTED	Callback aborted by user.
DCM_CANNOTGETSEQUENCEVALUE	Function failed to retrieve a value that had a representation of SQ.
DCM_ELEMENTCREATEFAILED	Fuction failed to create memory for a data element.
DCM_ELEMENTLENGTHERROR	Element length specified is longer than remaini data in stream or file input.
DCM_ELEMENTNOTFOUND	Function failed to find a data element requested by the caller.
DCM_ELEMENTOUTOFORDER	Data element not in numerically ascending order when reading data from a file or from a buffer in memory.
DCM_EXPORTINCOMPLETE	Export function is incomplete. More data in the object than will fit in the caller’s buffer.
DCM_NORMAL	Normal return from DCM function.
DCM_FILEACCESSERROR	Function failed to read a file after it was opened. Indicates the file does not contain syntactically correct data or that a file error occurred.
DCM_FILECREATEFAILED	Failed to create a file.
DCM_FILEDELETEFAILED	Failed delete a file.
DCM_FILEIOERROR	IO file when accessing (r/w) a file.
DCM_FILEOPENFAILED	Function failed to open requested file.
DCM_GETINCOMPLETE	DCM_GetElement value returned incomplete data because caller’s buffer was smaller than the length of the data element.
DCM_GROUPNOTFOUND	Could not find the group requested by caller.
DCM_ILLEGALADD	Caller attempted to add an illegal element to an existing object (such as a group length).
DCM_ILLEGALCONTEXT	Caller passed illegal context variable to DCM function.
DCM_ILLEGALOBJECT	Caller attempted function on an object that is not a legal DCM_OBJECT.
DCM_ILLEGALOPTION	Caller passed illegal option to DCM function.

```
DCM_ILLEGALREPRESENTATION  Illegal representation specified for data element.
DCM_ILLEGALSTREAMLENGTH    Function  encountered an illegal stream length
                           (negative or odd number).
DCM_INSERTFAILED           DCM function failed to insert new element as
                           requested.
DCM_LISTFAILURE            Failure in a list function caused DCM function
                           to fail.
DCM_MALLOCFAILURE          Failed allocate heap memory.
DCM_NORMAL                 Normal return from DCM function (success)
DCM_NULLADDRESS            Caller passed a NULL address to DCM function which
                           was expecting to write something back to caller.
DCM_NULLOBJECT             Caller passed NULL DCM_OBJECT to DCM function.
DCM_OBJECTCREATEFAILED     Function failed to create DCM_OBJECT.
DCM_UNEXPECTEDREPRESENTATION  Caller passed an element with an unexpected
                           value representation.  (It did not match the
                           VR in the internal dictionary.)
DCM_UNEVENELEMENTLENGTH    Uneven element length found in data element.
DCM_UNRECOGNIZEDELEMENT    Element not found in DCM dictionary
DCM_UNRECOGNIZEDGROUP      Group not found in DCM dictionary.
```

5 DCM Functions

This section contains detailed descriptions of the DCM functions.

DCM_AddElement

Name

DCM_AddElement - add a single data element to an existing information object.

Synopsis

```
CONDITION DCM_AddElement(DICOM_OBJECT **object,
                         DCM_ELEMENT *element)
```

object The existing DICOM information object.

element A description of a data element to be added to object.

Description

DCM_AddElement makes a copy of a data element passed by the caller and adds the data element to an existing information object. The caller is required to fill in the values for tag, length and data. The caller may optionally fill in representation and description (for a data element in a shadow group).

If the caller specifies representation as DCM_UNKNOWN, the function will look up the element in the DICOM dictionary before adding in to the object.

Adding a data element to an existing object involves several rules. These are enumerated below:

- Applications are not allowed to add Group Length elements (element 0000 in any group). The DCM facility maintains those elements and recalculates the group length when a data element is added to a group. In the event that a caller wishes to add a data element from a group that does not exist in the object, this function will automatically create the group and the corresponding Group Length element.
- Data elements can be added in any order. This function inserts the element in the proper location in the object.
- The caller can add an element which is not in the DICOM dictionary by specifiying the the representation DCM_UNKNOWN or DCM_OB. If the data is exported, the facility will not try to manage the data for the proper byte ordering. Thus, the application will have to maintain this data as private. This will be an issue if an application exports data to be read on a machine with a different architecture (byte order).

Return Values

DCM_NORMAL	DCM_NULLOBJECT
DCM_ILLEGALOBJECT	DCM_ILLEGALADD
DCM_LISTFAILURE	DCM_ELEMENTCREATEFAILED

Name

DCM_CloseObject - destroy the internal representation of an object and remove the caller's reference to the object.

Synopsis

CONDITION DCM_CloseObject(DCM_OBJECT **object)

object The caller's reference to the object to be closed.

Description

DCM_CloseObject closes an information object by destroying the internal representation of the object. After the object has been destroyed, the function removes the caller's reference to the object by writing NULL into the caller's handle. This function is called to close objects that have been created with *DCM_CreateObject*, *DCM_ImportStream* and *DCM_OpenFile*.

Return Values

```
DCM_NORMAL
DCM_NULLOBJECT
DCM_ILLEGALOBJECT
DCM_LISTFAILURE
```

Name

DCM_CreateObject - creates a new (empty) information object and returns a handle to the caller.

Synopsis

```
CONDITION DCM_CreateObject(DCM_OBJECT **object,
                           unsigned long opt)
```

object The handle for the object returned by this function.

opt Options argument which is used to determine features of the newly created object.

Description

DCM_CreateObject creates a new information object and returns a handle to the caller. The function allocates memory for the object and fills in the several fields that are hidden from the caller. After the object is created, the application can add data elements by calling *DCM_AddElement*.

The *opt* argument controls features of the newly created object. There is one feature that is implemented. The default mode is that objects will have group length elements for each group that is created. If the caller specifices DCM_NOGROUPLENGTH, group length elements will not be maintained and will note be exported if the caller invokes *DCM_ExportStream* or *DCM_WriteFile*.

Notes

DCM_OpenFile and *DCM_ImportStream* create information objects. Therefore, the caller should not call *DCM_CreateObject* and use the handle in a call to *DCM_OpenFile* or *DCM_ImportStream*.

Return Values

```
DCM_NORMAL
DCM_OBJECTCREATEFAILED
DCM_LISTFAILURE
```

Name

DCM_Debug - turn debugging messages on or off

Synopsis

```
void DCM_Debug(CTN BOOLEAN flag)
```

flag Indicates if debugging should be enabled.

Description

DCM_Debug is used to enable or disable debugging messages. The caller should pass TRUE or FALSE.

Return Values

None

Name

DCM_DumpElements - a developer's tool which dumps a description of each data element to the standard output.

Synopsis

```
CONDITION DCM_DumpElements(**object, long vm)
```

object Caller's handle to the information object to be dumped

vm Value multiplicity flag. Dump extra values for non zero vm

Description

DCM_DumpElements prints a short description of each data element in a object to the standard output. This description includes:

- tag value
- length
- english description

The function also prints a representation of data elements according to their representation (integer or ASCII string). The data printed may be incomplete because the output is limited to one line per data element.

The normal output for binary values is to print the first value of the attribute in hex and decimal. If the caller passes a non-zero *vm*, this function will print up to *vm* values of the attribute in decimal. This has been useful for dumping binary values in attributes like lookup tables.This will normally not work for pixel data because of the way that files are parsed.

Return Values

```
DCM_NORMAL
DCM_NULLOBJECT
DCM_ILLEGALOBJECT
```

Name

DCM_ElementDictionary - lookup one or more elements in the internal DCM dictionary

Synopsis

```
CONDITION DCM_ElementDictionary(DCM_TAG tag, void *ctx,
          void (*callback)(DCM_TAG t, char *description,
          DCM_VALUEREPRESENTATION r, void *ctx))
```

tag The DICOM tag that defines the attribute to be found in the dictionary.

ctx Caller context information to be supplied to caller's callback function.

callback Callback function invoked for each element found in the dictionary.

Description

DCM_ElementDictionary is used by callers to lookup one or more elements in the internal DCM element dictionary. The caller requests one element by passing a valid tag in tag; the caller can wildcard the group or element in the tag by using the value 0xffff as the number for the group or element.

For each element that matches the tag specificied, *DCM_ElementDictionary* invokes the callback function with the arguments shown in the prototype above. description is a description of the element with tag t. ctx is user context data that the caller may want to use during the callback function.

Notes

To get all of the elements in group 0x0028, use tag: DCM_MAKETAG(0x0028, 0xffff)

To get all elements, use tag: DCM_MAKETAG(0xffff, 0xffff)

To get all elements with element number 0x0004 regardless of group, use tag: DCM_MAKETAG(0xffff, 0x0004)

Return Values

DCM_NORMAL

Name

DCM_ExportStream - takes the internal representation of an information object and creates the stream representation suitable for output to a network or a file.

Synopsis

```
CONDITION DCM_ExportStream(DCM_OBJECT  **object,
          unsigned long options, void  *buffer,
          unsigned long length, CONDITION (*callback)(),
          void *ctx)
```

object	Caller's handle to the object to be exported.
options	Bitmask specifying options during export process (byte order).
buffer	Pointer to the caller's buffer to hold the exported data.
length	Length of the caller's buffer. Not necessarily large enough to hold the entire stream.
callback	User function which is called whenever DCM_ExportStream wants to dump output data.
ctx	Context variable provided by the caller and written by the function for successive calls to the function.

Description

DCM_ExportStream takes a DICOM object and creates a byte stream that conforms to a transfer syntax defined in Part 5 of the Standard. The caller specifies a buffer to hold the data and the length of the data. This function traverses the internal representation of the object and exports data into the caller's buffer. As the caller's buffer is filled, *DCM_ExportStream* calls the user callback function and expects that callback function to dispose of the byte stream that is created. Multiple calls to the user callback function may be necessary to complete the export operation.

The options argument is a bit mask defining options used during the export. Defined options are:

DCM_ORDERLITTLEENDIAN	Export the data in the little-endian format.
DCM_ORDERBIGENDIAN	Export the data in big-endian format.

When the user callback function is called, the arguments are:

buffer	Address of data exported by this function. This may or may not be the same as the buffer argument the user passed to this function.
bytes	An unsigned long which gives the number of bytes which are exported during this call.
last	An int flag which is TRUE (1) if this is the last buffer in the object and FALSE otherwise.
ctx	The caller's ctx argument. This provides the caller with a mechanism for maintaining context information in the callback.

The user callback function should return DCM_NORMAL if it successfully exports the data and DCM_EXPORTABORT if it wishes to halt the export process for any reason.

Return Values

DCM_NORMAL
DCM_NULLOBJECT
DCM_ILLEGALOBJECT
DCM_ILLEGALOPTION
DCM_EXPORTABORT

Code Example

```
static CONDITION callback(void *b, unsigned long l, int flag, in fd)
{

    if (flag)
        printf("Last buffer!\n");

    if (write(fd, b, l) == (int)l)
        return DCM_NORMAL;
    else {
        (void)close(fd);
        return DCM_EXPORTABORT;
    }

}

CONDITION writeObject(DCM_OBJECT *obj, char * name)
{
  CONDITION cond;
```

```
  unsigned char buf[2048];
  int fd;



    fd = open(name, O_CREAT | O_WRONLY, 0666);

    if (fd < 0)
        punt();

    cond = DCM_ExportStream(&obj, DCM_ORDERLITTLEENDIAN,buf,
                            sizeof(buf), callback, fd);

    if (cond != DCM_NORMAL)
        punt();

    return 1;

}
```

Name

DCM_GetElement - retrieve an element from an object minus the pointer to the data.

Synopsis

```
CONDITION DCM_GetElement(DCM_OBJECT **obj, DCM_TAG tag,
                         DCM_ELEMENT *attribute)
```

object The handle for the user's information object.

tag A tag (group, element) which identifies the element to be retrieved.

attribute Address of memory allocated by user to hold returned element.

Description

DCM_GetElement searches the caller's DICOM object for one attribute and returns a description of the attribute if it exists in the object. All fields in the attribute variable are filled in with the exception of the pointer to the actual data. The pointer is set to NULL to prevent the caller from modifying data that is maintained by the DCM facility.

Return Values

```
DCM_NORMAL
DCM_NULLOBJECT
DCM_ILLEGALOBJECT
DCM_ELEMENTNOTFOUND
```

Name

DCM_GetElementValue - retrieves the data for a single data element in a DICOM object..

Synopsis

```
CONDITION DCM_GetElementValue(DCM_OBJECT **object,
              DCM_ELEMENT *element,
              unsigned long *rtnLength, void **ctx)
```

object The handle for the user's information object.

element The description (tag) of the data element to be retrieved and a description of the location and size of the area to hold the data.

rtnLength The length of the data (in bytes) returned to the caller.

ctx Context variable used by this function for successive calls to retrieve data which does not fit entirely in the caller's buffer.

Description

DCM_GetElementValue is used to retrieve all or part of the data for a single data element. The caller identifies a data element by filling in the tag field in the element argument. The caller allocates memory to hold the data and specifies the address and length of the memory in the data and length fields of the element argument. The function copies as much data as will fit into the caller's area and writes the length of the returned data into the *rtnLength* argument. Subsequent calls to the function will continue copying data from the point where the previous call stopped. The caller should place NULL in the *ctx* argument before the first call to get data for a particular data element. The function uses this variable to maintain its position in the data stream.

As implied above, this function copies data to an area defined by the caller and does not allocate memory. This function does not terminate data elements which contain ASCII data. The function returns the length of the data and requires the user to terminate ASCII strings for use with standard run time libraries.

Notes

This function does not allow the caller to retrieve the value of an element that has a representation of DCM_SQ. Such elements contain a number of different types of values and do not fit the model of this function. User's should refer to the function

DCM_GetSequenceList for information on extracting data from elements that are sequences.

Return Values

```
DCM_NORMAL
DCM_ILLEGALOBJECT
DCM_GETINCOMPLETE
DCM_ELEMENTNOTFOUND
DCM_ILLEGALCONTEXT
DCM_CANNOTGETSEQUENCE
```

DCM_GetElementValueList

Name

DCM_GetElementValueList - create and return a list of string values from an element that may have a value multiplicity of greater than one.

Synopsis

```
CONDITION DCM_GetElementValueList(DCM_OBJECT **object,
                    DCM_TAG tag, size_t structureSize,
                    long stringOffset, LST_HEAD **list)
```

object Address of a pointer to a DCM_OBJECT. This is the handle for the object to be examined for the data element.

tag The 32-bit tag which identifies the data element of interest.

structureSize Size of the structure the caller wants to have placed in the list.

stringOffset Offset of the string value in the caller's structure.

list Address of a pointer to a LST_HEAD structure. This function will create a new list and write the pointer to the list in the caller's area.

Description

DCM_GetElementValueList is used to retrieve data from elements that are strings and may have a value multiplicity greater than one. Such values are encoded by concatenating them in a string with an explicit separator (\). This function parses the concatenated string and creates a structure for each string found. This structure is defined by the caller through the *structureSize* and *stringOffset* arguments. *stringOffset* tells this function where to place the ASCII string in the structure created by this function. Each structure that is created by this facility is placed in the caller's list by calling *LST_Enqueue*. The caller is responsible for creating the list before calling *DCM_GetElementListValue*.

Notes

This function does not attempt to clear the caller's list before adding new elements to the list. The caller may wish to use this feature to add items to a list that already contains data.

Return Values

DCM_NORMAL	DCM_NULLOBJECT
DCM_ILLEGALOBJECT	DCM_ELEMENTNOTFOUND
DCM_MALLOCFAILURE	DCM_LISTFAILURE

Name

DCM_GetElementSize - returns the length of the data for a single data element.

Synopsis

```
CONDITION DCM_GetElementSize(DCM_OBJECT  **object,
                  DCM_TAG tag, unsigned long *rtnLength)
```

object The caller's handle for the information object.

tag The tag value (group, element) which uniquely defines the attribute (as defined in the data dictionary in Part 6 or in the Annex of other parts of the Standard).

rtnLength Pointer to caller's variable to hold returned length.

Description

DCM_GetElementSize searches the information object identified by object for the data element defined by tag. The function returns the length of the data by writing it into the caller's rtnLength variable.

If the requested element is not found in the caller's object, *DCM_GetElementSize* does not update the caller's rtnLength variable and returns DCM_ELEMENTNOTFOUND.

Return Values

```
DCM_NORMAL
DCM_NULLOBJECT
DCM_NULLOBJECT
DCM_ILLEGALOBJECT
DCM_ELEMENTNOTFOUND
```

DCM_GetObjectSize

Name

DCM_GetObjectSize - detemines the length of an information object when represented in stream format and returns the length to the caller.

Synopsis

```
CONDITION DCM_GetObjectSize(DCM_OBJECT  **object,
              unsigned long *rtnLength)
```

object The caller's handle to the information object.

rtnLength Pointer to the caller's variable to hold length of the object.

Description

The DCM facility is designed to support information objects in stream format. *DCM_GetObjectSize* determines the length of an object when represented in stream format and returns that length to the caller.

Return Values

```
DCM_NORMAL
DCM_NULLOBJECT
DCM_NULLOBJECT
DCM_ILLEGALOBJECT
```

DCM_GetSequenceList

Name

DCM_GetSequenceList - return the head of the list of a set of items in a sequence.

Synopsis

```
CONDITION DCM_GetSequenceList(DCM_OBJECT **object,
                              DCM_TAG tag, LST_HEAD **list)
```

object Address of a pointer to a DCM_OBJECT. This is the handle for the object to be examined for the data element.

tag The 32-bit tag which identifies the data element of interest.

list Address of a pointer to a LST_HEAD object. This function will store the LST_HEAD object which contains the sequence items at this address.

Description

DCM_GetSequenceList searches an information object for an element specified by the caller's tag argument. This element is assumed to have a representation of DCM_SQ. If the element is found and has the proper representation, this function stores the LST_HEAD object of the sequence at the address specified by the caller's list argument.

This function allows the caller to get the list of DCM_OBJECTS that are in a sequence. The caller may then examine the individual items in the list.

Notes

Because this function returns a copy of the LST_HEAD pointer, the caller must not modify any of the objects in the list. The items are still maintained by the original object.

Return Values

```
DCM_NORMAL
DCM_NULLOBJECT
DCM_ILLEGALOBJECT
DCM_ELEMENTNOTFOUND
```

DCM_GroupDictionary

Name

DCM_GroupDictionary - lookup one or more groups in the internal group dictionary.

Synopsis

```
CONDITION DCM_GroupDictionary(unsigned short group,
                    void *ctx,
                    void (*callback)(unsigned short g,
                       char *description, void *ctx))
```

group The group the caller would like to lookup in the DCM dictionary.

ctx Caller context information to be supplied to caller's callback function.

callback Callback function invoked for each group found in the dictionary.

Description

DCM_GroupDictionary is used by callers to lookup one or more groups in the internal DCM group dictionary. The caller requests one group by passing a valid group number in group; the caller specifies all groups by passing the value 0xffff in group.

For each group that matches (one or all), *DCM_GroupDictionary* invokes the callback function with the arguments shown in the prototype above. *description* is a description of the group with number *g*. *ctx* is user context data that the caller may want to use during the callback function.

Return Values

DCM_NORMAL

Name

DCM_ImportStream - imports a stream of data in DICOM V3 format and creates an information object to represent the data.

Synopsis

```
CONDITION DCM_ImportStream(void *buf, unsigned long length,
              unsigned long options, DCM_OBJECT **object)
```

buf Pointer to caller's buffer holding stream data to be imported.

length Length of the data stream in bytes.

options Bitmask giving options to function during import process.

object Handle to information object to be created and returned to caller.

Description

DCM_ImportStream creates a new information object and fills it by passing the data (buf, length) supplied by the caller. The data is assumed to comply with Part 5 of the DICOM V3 Standard. The Implicit Little-Endian transfer syntax is implemented as well as data in big-endian format. The big-endian data format supports image data supplied by vendors with V2 data and does not comply with the Explicit Big-Endian transfer syntax defined in Part 5. The function performs minor rules checking to determine if the stream complies with the standard.

The *options* argument is a bitmask which specifies options for interpreting the data stream as it is passed. The defined options are:

DCM_ORDERLITTLEENDIAN
: Use little-endian format to interpret the data.

DCM_ORDERBIGENDIAN
: Use big-endian format to interpret the data.

DCM_FORMATCONVERSION
: Convert format as data is imported. This includes stripping leading and trailing blanks converting some old V2 formats to new V3 formats.

DCM_NOGROUPLENGTH
: Remove group length elements if they exist in the input stream.

Return Values

```
DCM_NORMAL
DCM_ILLEGALOPTION
DCM_OBJECTCREATEFAILED
DCM_ELEMENTOUTOFORDER
DCM_LISTFAILURE
```

DCM_IsString

Name

DCM_IsString - Determine if a DICOM Value Representation is a string type.

Synopsis

CTNBOOLEAN DCM_IsString(DCM_VALUEREPRESENTATION representation)

representation One of the enumerated value representations defined in the include file for this facility.

Description

DCM_IsString examines the representation argument passed by the caller and determines if the DCM facility considers the data to be in string format. If so, this implies that the caller will be able to use some of the standard C run-time libraries on the data given that the data is properly terminated.

The set of value representations that are considered to be strings are:

DCM_AE	DCM_IS	DCM_ST
DCM_AS	DCM_LO	DCM_TM
DCM_CS	DCM_LT	DCM_UI
DCM_DA	DCM_PN	
DCM_DS	DCM_ST	

Return Values

TRUE
FALSE

Name

DCM_ListToString - turn a list of ASCII values into a single string with DICOM separators.

Synopsis

```
CONDITION DCM_ListToString(LST_HEAD *list, long offset,
                           char **string)
```

list Caller's list which contains an ordered set of ASCII strings to be concatenated into a single string.

offset Offset (in bytes) of the ASCII data in the items stored in the caller's list.

string Address of a pointer to a character string. This function will allocate memory to hold the output string and will write the address of the allocated memory in the caller's string pointer.

Description

DCM_ListToString is used to support string-type data elements with value multiplicity greater than 1. This function assumes the caller has a homogenous set of items which are stored in a list (maintained by the LST facility). The ASCII data is expected to be found offset bytes from the beginning of these items and is terminated with a NULL in all cases. *DCM_ListToString* examines the caller's list and determines the total number of bytes required to store a concatenation of these strings and allocates enough memory to do so. *DCM_ListToString* then extracts the ASCII data in each item and places it in the allocated memory. String items are separated with the standard DICOM separator (\). A NULL terminator is written on the tail of the string, allowing the string to be manipulated with standard C run-time libraries.

It is the caller's responsibility to free the memory allocated for the string.

Notes

This is a dangerous function because it can lead to core dumps if the function is called with the wrong offset or with improperly terminate data. It is also prone to memory leaks if the caller is not careful to free allocated memory when finished.

Return Values

```
DCM_NORMAL
DCM_MALLOCFAILURE
```

Name

DCM_LookupElement - find information about an element in the data dictionary.

Synopsis

```
CONDITION DCM_LookupElement(DCM_ELEMENT *element)
```

element Address of structure in caller's address space which contains the tag (group, element) defining the element and memory to hold information found in the dictionary.

Description

DCM_LookupElement searches the data dictionary for a single data element using the tag value passed by the caller in the element argument. If the element is found, the function fills in the values for *representation* and *englishDescription*.

Return Values

```
DCM_NORMAL
DCM_UNRECOGNIZEDGROUP
DCM_UNRECOGNIZEDELEMENT
```

Name

DCM_ModifyElements - modify or add one or more elements in a DICOM Information Object.

Synopsis

```
CONDITION DCM_ModifyElements(DCM_OBJECT **object
              DCM_ELEMENT *requiredList, int count,
              DCM_FLAGGED_ELEMENT *flaggedList,
              int flagCount, int *updateCount)
```

object Handle to the caller's object to be modified.

requiredList Pointer to an array of elements that must be modified.

count Number of elements in the array of required elements.

flaggedList Pointer to an array of elements that may be optionally modified.

flagcount Number of elements in the array of optional elements.

updateCount Number of elements that were successfully modified.

Description

DCM_ModifyElements modifies existing elements in a DICOM Information Object or adds new elements if they do not currently exist. The caller supplies two arrays of elements. The elements in the array *requiredList* are of type DCM_ELEMENT and describe attributes that will always be modified. The elements in the array *flaggedList* are of type DCM_FLAGGED_ELEMENT and are conditionally modified.

The DCM_FLAGGED_ELEMENT structure contains a bit mask and a pointer to a flag. An element of this type is updated only if the bit mask in the element is set at the address pointed at by the flag value.

These routines are typically used to copy values from a fixed structure to a DICOM Information Object. Some elements are always present in the structure (mandatory), and can be described using the *requiredList* argument. Other elements may be conditional. The *flaggedList* describes the conditional elements and provides bit masks for indicating which of the elements should be updated.

This function modifies as many elements as possible and writes the number of successfully modified elements in the variable pointed at by *updateCount*. If the caller passes a NULL pointer, no count is returned.

Return Values

DCM_NORMAL

Code Example

The code example below modifies the number of rows and the window center value in an existing Information Object. The number of rows is always updated and is set to 1024. The window center value is updated only if the ASCII string in the structure im is not of 0 length. In the example below, the ASCII string is set to “100”, so the code will update the window center value.

```
#defineCENTER_BIT     0x01;
  typedef struct {
    long        flag;
    unsigned short rows;
    char        center[24];
} IM;

  static IM im = { 0, 1024, “100” };

  DCM_ELEMENT r = { DCM_IMGROWS, DCM_US, “”, 1,
                    sizeof(im.rows), (void *)&im.rows };

  DCM_FLAGGED_ELEMENT w = DCM_IMGWINDOWCENTER, DCM_DS, “”, 1,
                    0,(void *)im.center, CENTER_BIT, &im.flag };

  im.flag = 0;

  w.e.length = strlen(w.e.d.string);

  if (w.e.length != 0)

  im.flag |= CENTER_BIT;

  cond = DCM_ModifyElements(&object, &r, 1, &w, 1, NULL);
```

Name

DCM_OpenFile - opens a file containing a DICOM V3 object and creates an in-memory representation of the object.

Synopsis

```
CONDITION DCM_OpenFile(char *name, unsigned long options,
          DCM_OBJECT **object)
```

name Full or relative path name of the file to be opened.

options Bitmask specifying options to be used when file is opened and read.

object The handle to the object created by this function and returned to the caller.

Description

DCM_OpenFile opens a file (*name*) that contains a DICOM V3 compliant object. The file is opened for read only access, the DICOM stream is parsed and an in-memory representation of the object is created. The function creates a handle which is used to reference the object and returns the handle to the caller through the argument object.

The options argument is used to control the file open and passing of the data stream. The caller should (mathematically) OR constants found in the DICOM OBJECT include file to create a legal set of options. Recognized options are:

DCM_ORDERLITTLEENDIAN
: Use little-endian format to interpret the data.

DCM_ORDERBIGENDIAN
: Use big-endian format to interpret the data.

DCM_FORMATCONVERSION
: Convert format as data is imported. This includes stripping leading and trailing blanks converting some old V2 formats to new V3 formats.

DCM_NOGROUPLENGTH
: Remove group length elements if they exist in the input stream.

DCM_PART10FILE
: File is a DICOM Part 10 compliant file

Return Values

```
DCM_NORMAL
DCM_ILLEGALOPTION
DCM_CREATEOBJECTFAILED
DCM_LISTFAILURE
DCM_FILEOPENFAILED
DCM_FILEACCESSERROR
DCM_ELEMENTOUTOFORDER
```

Name

DCM_ParseObject - examine the contents of an Information Object and extract the data from one or more elements.

Synopsis

```
CONDITION DCM_ParseObject(DCM_OBJECT **object,
          DCM_ELEMENT *requiredList, int count,
          DCM_FLAGGED_ELEMENT *flaggedList,
          intflagCount, int *parseCount)
```

object Handle to the caller's object to be parsed.

requiredList Address of an array of elements that are required to be in the object. Failure to find any of the elements in this array will cause a failure.

count Number of elements in the required list.

f*laggedList* Address of an array of elements that may be in the object but are not required. The function will search for these elements and set bit masks for any elements that are found.

flagCount Number of elements in the list of optional elements.

parseCount Address of caller's variable to hold the number of elements found by this function.

Description

DCM_ParseObject examines an Information Object for a list of required and optional elements. For each element that is found, *DCM_ParseObject* extracts the data and places it at the address which has been specifed by the caller. The caller is expected to have allocated memory and described that memory in the DCM_ELEMENTs found in *requiredList* and *flaggedList*. Any attribute that has a value representation that is an ASCII string is terminated with a 0 (for later use with standard C run-time libraries).

DCM_ParseObject looks for the elements in requiredList. If any element in that list is not found, the function fails and returns immediately. After searching for the required elements, the function searches for the optional elements in *flaggedList*. Elements which are not found are ignored. For each element that is found, *DCM_ParseObject* will set a bit in a

flag variable as described in the DCM_FLAGGED_ELEMENT passed through *flaggedList*.

This function maintains the number of successfully parsed elements and places the count at the integer whose address is *parseCount*. If the caller passes NULL for this argument, the count is not returned to the caller.

Notes

This function calls *DCM_GetElementValue*. It assumes that the caller has allocated enough space to hold each of the elements. This function will return values returned by *DCM_GetElementValue* if that function is not able to successfully retrieve a value fromt the requiredList.

Return Values

DCM_NORMAL
DCM_NULLOBJECT
DCM_ILLEGAL OBJECT
DCM_ELEMENTNOTFOUND
DCM_CANNOTGETSEQUENCE
DCM_FILEACCESSERROR
DCM_GETINCOMPLETE

Code Example

The code example below parses an object and looks for the number of rows and the window center value. After the call to DCM_ParseObject, we can tell if the object had the WINDOW CENTER attribute by testing a bit mask.

```
#defineCENTER_BIT     0x01;

  typedef struct {
    long        flag;
    unsigned short rows;
    char        center[24];
} IM;

  static IM im;
  DCM_ELEMENT r = { DCM_IMGROWS, DCM_US, “”, 1, sizeof(im.rows),
                    (void *)&im.rows };
  DCM_FLAGGED_ELEMENT w = { DCM_IMGWINDOWCENTER, DCM_DS, “”, 1, 0,
                            (void *)im.center, CENTER_BIT, &im.flag  };

  im.flag = 0;
  cond = DCM_ParseObject(&object, &r, 1, &w, 1, NULL);
  if (im.flag & CENTER_BIT)
/* Object had a WINDOW CENTER value */
```

Name

DCM_RemoveElement - remove an element from an information object.

Synopsis

```
DCM_RemoveElement(DCM_OBJECT **object, DCM_TAG tag)
```

object The caller's handle to the information object.

tag The tag (group, element) which defines the element to be removed.

Description

DCM_RemoveElement removes a single element from an information Object. Any memory allocated for the internal representation of that element is destroyed.

Return Values

```
DCM_NORMAL
DCM_ILLEGALOBJECT
DCM_NULLOBJECT
DCM_GROUPNOTFOUND
```

Name

DCM_RemoveGroup - remove an entire group from a DICOM Information Object.

Synopsis

```
CONDITION DCM_RemoveGroup(DCM_OBJECT **object,
                  unsigned short group)
```

object The caller's handle to the information object.

group The number of the group to be removed.

Description

DCM_RemoveGroup removes a single group and its collected attributes from an Information Object. Any memory allocated for the internal representation of the individual attributes or group structure is destroyed.

Return Values

```
DCM_NORMAL
DCM_ILLEGALOBJECT
DCM_NULLOBJECT
DCM_GROUPNOTFOUND
```

DCM_ScanParseObject

Name

Synopsis

Description

Notes

Return Values

Name

DCM_WriteFile - Write an encoded DICOM Information Object to a file.

Synopsis

```
CONDITION DCM_WriteFile(DCM_OBJECT **object,
          unsigned long options, char *file)
```

object Address of caller's pointer to a DICOM Object. This object is exported to a file.

options Flag containing options which control how the object is encoded when written to the file.

file Name of file to be created and used for output.

Description

DCM_WriteFile exports an encoded version of an Information Object (stream format) and writes the data to a file. The user supplies a handle for an Information Object and a set of flags which control how the data are exported to the file. This function creates a new file with name file and writes the stream representation of the data to that file.

Recognized options are:

DCM_ORDERLITTLEENDIAN Use little-endian format to interpret the data.

DCM_ORDERBIGENDIAN Use big-endian format to interpret the data.

Return Values

```
DCM_NORMAL
DCM_NULLOBJECT
DCM_ILLEGALOBJECT
DCM_FILECREATEFAILED
DCM_FILEIOERROR
DCM_ILLEGALOPTION
DCM_LISTFAILURE
```