

# DicomObjects User Manual

## Contents

<b>1</b>	<b>Introduction.....</b>	<b>3</b>
1.1	Structure.....	3
1.2	Languages Supported.....	4
1.3	Effects of Using the COM Object Model.....	4
1.4	Collections in DicomObjects.....	4
1.5	DicomObjects' Representation of DICOM Data.....	5
1.6	Interpreting and Using Sequences.....	7
1.7	Private Attributes.....	7
<b>2</b>	<b>First steps - Reading, Viewing and Writing DICOM files.....</b>	<b>9</b>
2.1	Your First DICOM Program.....	9
2.2	Writing an Image to Disk.....	10
<b>3</b>	<b>Simple Sending and Receipt of images over a network.....</b>	<b>11</b>
3.1	Sending an Image.....	11
3.2	Receiving Images.....	11
<b>4</b>	<b>Query/Retrieve (SCU).....</b>	<b>13</b>
4.1	Common Features.....	13
4.2	DoQuery.....	14
4.3	GetImages.....	14
4.4	GetUsingMove.....	15
4.5	DoRawQuery.....	15
4.6	MoveSync.....	15
4.7	MoveImages.....	15
4.8	DicomConnection Based Q/R Methods.....	15
<b>5</b>	<b>Off-line Media.....</b>	<b>17</b>
5.1	Reading.....	17
5.2	Creating.....	18
5.3	Updating.....	19
5.4	Multiply Referenced Directory Records.....	19
<b>6</b>	<b>Printing.....</b>	<b>20</b>
6.1	Printing Using DicomPrint.....	20
6.2	Printing Using Normalised Operations.....	22
6.3	Printing DICOM Images to a Windows Printer.....	22
<b>7</b>	<b>Exporting DICOM Images to Other Formats.....</b>	<b>23</b>
7.1	Single Frames.....	23
7.2	Multi-frame Images/Cine.....	23
7.3	Non-file Exporting.....	23
<b>8</b>	<b>Advanced Image Review station.....</b>	<b>24</b>
8.1	Basic Viewing Controls.....	24
8.2	Multi-frame (Cine) Images.....	24
8.3	Annotations.....	25
8.4	Lookup Tables.....	28
8.5	DICOM Greyscale Presentation State.....	28
8.6	Display Speed Optimisation.....	29
<b>9</b>	<b>Web Usage.....</b>	<b>31</b>

9.1	Running DicomObjects on a Web Server .....	31
9.2	Running DicomObjects on the web client .....	33
<b>10</b>	<b>Writing a Router/Modifier .....</b>	<b>35</b>
<b>11</b>	<b>Writing a DICOM Server .....</b>	<b>36</b>
11.1	An object to Listen for Associations .....	36
11.2	Validating Associations .....	37
11.3	Handling C-STORE Operations .....	37
11.4	Handling Query/Retrieve Requests .....	38
11.5	Handling C-ECHO Requests .....	42
11.6	Transfer Syntax and Quality Issues .....	42
11.7	Performance and Reliability Issues .....	43
11.8	Modality WorkList SCP .....	44
11.9	Print SCP .....	44
11.10	Storage Commitment SCP .....	45
<b>12</b>	<b>Accessing and Modifying Pixel Data .....</b>	<b>46</b>
12.1	Languages with Raw Pointers .....	46
12.2	Languages Using Variant Arrays .....	47
<b>13</b>	<b>Creating DICOM Images .....</b>	<b>48</b>
13.1	Importing Other Formats .....	48
13.2	Importing Multiframe images .....	49
13.3	From Scratch .....	50
<b>14</b>	<b>Using Modality WorkList as an SCU .....</b>	<b>52</b>
<b>15</b>	<b>Language Specific Features .....</b>	<b>53</b>
15.1	Visual Basic .....	53
15.2	VBScript .....	53
15.3	Visual Basic for Applications (e.g. MS Access) .....	53
15.4	Microsoft Visual C++ .....	53
15.5	Borland Delphi & Borland C++ Builder .....	55
15.6	Java .....	56
15.7	Other Environments .....	56
<b>16</b>	<b>Logging .....</b>	<b>57</b>
16.1	Log Details and Levels .....	57
16.2	File Logging .....	57
16.3	The DicomLog Control .....	58
<b>17</b>	<b>Advanced Usage .....</b>	<b>59</b>
17.1	Over-riding Registry Values .....	59
17.2	Altering the List of Default SOP Classes .....	59
17.3	Transfer Syntax Selection .....	60
17.4	Private SOP classes .....	61
17.5	Private Transfer Syntaxes .....	61
17.6	Storage Commitment .....	61

# 1 Introduction

This user manual is an important complement to the DicomObjects help file. The help file remains the primary reference guide for all DicomObjects objects, methods and properties, so if you know which you need, go directly to the help file. This manual however, starts from the opposite point of view, and tells you how to use DicomObjects to accomplish specific aims. In many cases you will still need to reference the help file for full details of parameters etc., but at least after reading this manual, you should know which sections you need to consult!

The purpose of DicomObjects is to allow you, the application developer, to use DICOM in your applications, with as little need as possible for knowledge of the intricacies of the DICOM standard. It is of course not the only DICOM “toolkit” available, but the hope is that it fills an important niche in the market, due to the following features:

- DICOM is accessed through high-level interfaces, allowing many operations to occur on whole images etc.
- Finer level details are available where needed, allowing modification of individual image attributes.
- **All** DICOM SOP classes are supported, even types of image or other data not yet defined.
- Internal multi-threading, allows high-performance servers to be constructed even using single-threaded clients such as Visual Basic.
- A high level of support and advice is provided as expected from a commercial product.
- The pricing structure removes the “up-front” costs often associated with such toolkits.

There are many possible uses for DicomObjects, but here is a brief list of some common ones:

- Image viewing software (either stand-alone or web-based)
- DICOM image servers
- DICOM image creation from scratch in radiological equipment
- Import and export of images from and to other formats
- Teleradiology applications
- Intermediate routing stations, receiving images from one application and sending to another, modifying the data itself, or the surrounding associations to overcome incompatibilities.
- Modality worklist servers, using data from ODBC or HL7
- Quality assurance
- Off-line media readers and writers including DICOMDIR handling
- DICOM and Windows printing
- Diagnostic test applications

## 1.1 Structure

DicomObjects uses only a small selection of objects, but most of these have a wide variety of methods and properties available. The main items are:

- **DicomViewer**  
This is one of only two objects to occupy space in a window at runtime, and can display multiple images.
- **DicomImage and DicomDataSet**  
These two hold DICOM instances, and have many similarities, the difference being that a DicomImage has pixel data, whereas a DicomDataSet normally does not.

- **DicomAttribute**  
This represents a single attribute (e.g. study date) within one of the above objects.
- **DicomPrint**  
An object to simplify DICOM printing.
- **Collections**  
DicomImages, DicomDataSets and DicomAttributes objects hold multiple items of the corresponding types.
- **Advanced Objects**  
Other objects include the DicomLog, DicomServer, DicomGlobal and DicomContext(s), but these will be introduced as necessary below.

## 1.2 Languages Supported

The language used for all examples in this manual is Visual Basic. There are several reasons for this, the main ones being:

- It is the language used within Medical Connections for client programming.
- The structure is fairly obvious even to those not used to it.
- Using one consistent language throughout keeps this manual simple.

Of course, the language specific sections use appropriate code!

## 1.3 Effects of Using the COM Object Model

DicomObjects uses the Windows COM model exclusively, which has the following advantages:

- Language independence
- Encapsulation within a replaceable file (The “OCX” file)

However, there are a few drawbacks and common mistakes:

- Every method must apply to one or more objects, as there are no “freestanding” methods. An example where this can be confusing is the “ReadFile” method, which has to be applied to an existing object (a DicomImages collection or DicomGlobal object) to create a new object of a different type.
- The DicomObjects file must be distributed and registered with every copy of the application. Fortunately, with version 4, other file dependencies have now virtually been eliminated, and most modern installers have no problems adding and registering components such as DicomObjects.

## 1.4 Collections in DicomObjects

There are a number of “Collection” objects in DicomObjects, which share many common methods, enabling objects to be added to, accessed or removed from the collection. The following properties and methods are common to all collections in DicomObjects.

- **Count**  
The number of items currently in the collection
- **Item(long integer n)**  
Returns a reference to the n<sup>th</sup> item in the collection. This is the default member of the collection, so for instance Images.Item(n) may be replaced in Visual Basic by Images(n). Not all languages support this simplification however, so the full version or other forms such as Images.GetItem(n) may be necessary in some environments.

- **Add (Object)**  
Adds an object of an appropriate type to the end of the collection. In general (but with a few important exceptions), only objects of the appropriate type may be added to a collection, e.g. only DicomLabel objects may be added to a DicomLabels collection.
- **Remove (long integer n)**  
Removes item n from the collection, moving items above it down to fill the gap.
- **Clear**  
Completely empties the collection, removing all items from it.

**Notes:**

- In DicomObjects, all collection indices start from 1 (not 0), so the range of items is from Item(1) to Item(Count).
- A DicomContexts collection is indexed by presentation context ID, which must in DICOM always be a small odd integer, so indices are 1, 3, 5 etc.
- DicomImages and DicomDataSets collections allow indexing by instance UID, so the index is actually a variant, which should be either an integer, or a UID string.

### 1.5 DicomObjects' Representation of DICOM Data

The object most commonly used in DicomObjects is the DicomImage, which encapsulates a single SOP instance, normally including pixel data. A DicomDataSet is virtually the same, but is normally used for datasets (e.g. query results or elements within sequences) which do not contain pixel data. For the remainder of this section, a DicomImage (or just *image*) will be used for examples, but unless specified otherwise most non-pixel related features apply equally to both.

An important concept to grasp early on is that a DicomImage includes two completely separate sources of data:

- The original DICOM dataset
- Ephemeral data such as zoom factor, orientation, associated annotations and windowing values

In general, the DICOM data itself is accessed and modified through the image's Attributes property, and ephemeral data through simple properties, but there are a few properties such as Name, PatientID etc., which are in fact just shortcuts to the underlying DICOM data. All such properties are clearly indicated as such in the help file. A few properties, such as window Width and Level are ephemeral properties which may be altered without modifying the underlying data, but where possible, they are initialised using data from the DICOM dataset.

There are 2 important and often misunderstood consequences of the separation of the two types of data in a DicomImage:

- Two or more DicomImages can share the same permanent data, yet have their own ephemeral data. This facility is in fact used whenever a DicomImage is added to a DicomImages collection, as this method causes a completely new DicomImage to be created and added to the collection. This new image references the same permanent data, and is initialised with a copy of the original ephemeral data, but this may subsequently be modified. This procedure saves memory for the large DICOM dataset, yet allows different display parameters (e.g. "dual window" displays).
- Changes made to simple DicomImages properties do not generally modify the underlying DICOM data (other than for the "shortcut" properties mentioned above). So, if it is required to save the current window and width settings with an image, this must be done explicitly by adding Attributes 0x0028,0x1051 & 0x0028,0x1052, as described below.

References to the DICOM dataset held in an image are made through its `Attributes` property, which is a collection object. Slightly different procedures apply for accessing (reading) and modifying the data.

### 1.5.1 Accessing Attributes

An individual attribute is normally accessed from any language using the group and element tag numbers. In some languages a syntax such as `image.Attributes.Item(gggg,eeee)` or `image.Attributes.GetItem(gggg,eeee)` is required. However, as `Item` is the default property for a `DicomAttributes` object, many languages allow a syntax more like `image.Attributes(gggg,eeee)`. The value returned by this property is a `DicomAttribute`, which has a few properties, but the main ones of interest are:

- **Exists:** True if the item is present in the dataset.
- **Value:** A variant representation of the data contained  
This is the default property, and the explicit name can in some languages be omitted.

The type of the variant generally reflects the type of the underlying DICOM data element, but there are a few pitfalls to consider:

- If the data element is absent (`Exists=false`), then the type is `VT_ERROR`
- If the data is of zero length (common for type 2 elements and in query datasets) the type is `VT_NULL`
- Pixel data (element `0x7FE0,0x0010`) is handled differently, and returns the same value as the `Pixels` property – i.e. a 3 dimensional array of bytes or short integers.
- If the element is a sequence, the value is a `DicomDataSets` object (`VT_DISPATCH`). See below for details of sequence handling.
- Otherwise, if the data element could possibly contain multiple values (i.e. the DICOM VM is not “1”), then the variant returned will be a single dimensional array of the appropriate type of value, even if only a single item happens to be present. This will also apply to any attributes unknown to `DicomObjects` such as private elements, as the VM cannot be determined. The rationale for this behaviour is to simplify coding, as a programmer does not need to write separate code for single and multiple value cases.
- As of version 4.1 it is no longer necessary to handle multiple values using a variant array, as the number of items may be determined using a `DicomAttribute`'s “VM” property, and the individual items may then be accessed directly using the `ValueByIndex(n)` property.

For some applications it is necessary to iterate through all the attributes, without necessarily knowing in advance which tags are present. In languages which support direct iterators on collections, this is easy (e.g. “For Each” in Visual Basic), but for languages which lack this facility, the `ItemByIndex` method is provided.

Some scripting languages (notably VBScript, and some unusual scripts such as that used in MatLab), cannot use arrays consisting of anything except variants. For these environments, the `VariantValue` is provided, which can return variant arrays of variants – see the help file for more details.

Another variation on the `Value` property is the `UnicodeValue`. If the value is a suitable string, and if the character set attribute (`0x0008,0x0005`) is present, then the value is interpreted according to the appropriate character set rules (if installed on the operating system!).

### 1.5.2 Changing the DICOM data

As we cannot know in advance whether a particular attribute will even exist in a dataset, `DicomObjects` does not allow direct modification of the value of a `DicomAttribute`. Instead, particular members of a `DicomAttributes` collection may be replaced or removed, using the `Add` and `Remove` methods

respectively. The Add method uses a variant parameter to hold the value, which should normally be of the same type as listed above for the Attribute.Value result, with the following modifications:

- Values will be coerced as necessary – e.g. the value “2” would be valid for any numerical VR.
- A zero-length string may be used to set null value.
- Single values may be used even when the VM is “1-n” (though arrays are also of course allowable).

Where an attribute with the same tags already exists, it is simply replaced.

## 1.6 Interpreting and Using Sequences

DICOM allows one attribute to contain a sequence of datasets, and DicomObjects provides direct support for this arrangement by allowing one attribute to contain a DicomDataSets object, which in turn may contain one or more DicomDataSet objects. This structure is inherently recursive, and nesting to any depth is allowed (some DICOM IODs require 4 level nesting!), and can be used for both creation and read access to data in sequences. Note though that recursion is through two levels, as a DicomDataSet object contain a DicomDataSets collection, which contains one or more DicomDataSet objects, so it is necessary to be careful with the syntax. The full syntax for retrieving the first item of an embedded sequence would be:

```
Set sequenceitem=dataset.Attributes(gggg,eeee).Value.Item(1)
```

In many languages this can be contracted using the default properties to something like:

```
Set sequenceitem=dataset.Attributes(gggg,eeee).Item(1)
```

Further contraction can cause problems (e.g. in Visual Basic), as function parameters and indexes can be confused, so a common error is to attempt:

```
Set sequenceitem=dataset.Attributes(gggg,eeee).Value(1)
```

Of course, where a variable is used to represent the DicomDataSets collection, this confusion does not arise.

To add sequences to a dataset, first construct the new DicomDataSets collection, create new DicomDataSet items to add to it (using either stand-alone creation or the AddNew method), and then finally add the collection to the top level dataset. e.g.

```
Set sequence=new DicomDataSets
Set item=sequence.AddNew
Item.Attributes.Add(...)
Toplevel.Attributes.Add gggg,eeee, sequence
```

In some languages, sequence may need to be cast explicitly into a variant of type VT\_DISPATCH.

## 1.7 Private Attributes

DICOM allows programmers to create their own “private” attributes for internal uses, which should always have an odd group number, but this creates a problem for DicomObjects, as it does not know in advance what their value representation (VR) will be. DicomObjects has an internal dictionary of all officially known tags at its release date, but new tags are being added all the time, and a programmer using the latest definitions may face the same problem. DicomObjects provides two ways of dealing with this problem:

### 1.7.1 AddExplicit

This method is a minor variant on the normal Attributes.Add method, supplying a two character VR code, in addition to the tag numbers and the value. This code must be one of the standard VR types such as “US” (unsigned short), “LO” (long text) etc.

## **1.7.2 AddToDictionary**

This is a method of the DicomGlobal object, and adds any attribute you like (whether private or new) to the internal DicomObjects dictionary. This must happen each time the program is run, as only the copy of the table in RAM is modified, but it does have a few advantages over AddExplicit:

- A Description can also be added (later retrievable using DicomAttribute.Description)
- The VR applies to all subsequent use, including reading/receipt via an implicit VR transfer syntax.
- A VM may be added, which improves error checking, and allows the DicomAttribute.Value property to return single values where appropriate, rather than arrays.

A few other points about unknown/private elements are worth noting:

- DicomObjects properly uses the VR “UN” for unknown items received via an implicit VR transfer syntax
- If an attribute has type “UN”, either because of implicit receipt, or because it was explicitly sent/written as UN, you may, if you know the type, change it by setting the Attribute’s VR property directly (this was read-only prior to version 4)
- All private elements should be accompanied by an element gggg,0x0010, which is always of type LO (long string), describing in rough terms what the Attributes are (e.g. “XYZ Company Internal Values”)

## 2 First steps - Reading, Viewing and Writing DICOM files

Although DICOM was first used for network communications, and many developers will be using it for that purpose, DICOM files are used for this introduction to DICOM, as they are conceptually easier, and there is no reliance on external equipment.

### 2.1 Your First DICOM Program

Basic examples are provided in most of the major languages, but here is a recipe for producing a minimalist image viewer from scratch. As in all examples, the actual script is visual Basic, but the equivalent should be clear in any other environment.

- Import DicomObjects into your development environment
- Create a new form/dialog box
- Add a DicomViewer to that form (if your GUI uses icons for this, it will be shown as ). Rename this object to “Viewer”
- Add a command which performs the following:  

```
Viewer.Images.ReadFile
"C:\Program Files\Medical Connections\DicomObjects\Examples41\
  ↳Images\mono.dcm"
```

 [Modify path if installed to non-default location]
- Run the program, and activate the command you have created.

If all is well, an obstetric ultrasound image should be shown (my twins!), scaled to fit the DicomViewer you have created.

So what does the above command actually do, and how does this introduce you to use of DicomObjects? In fact, you have used 3 of the 10 different object types available, a DicomViewer, which provides a display area within a window, a DicomImage, which represents a single DICOM image, and a DicomImages collection, which as expected, holds a collection of DicomImage objects. We will consider each of these in turn:

#### 2.1.1 The DicomViewer

This provides a rectangular area within which one or more images may be displayed. It has many properties and methods, to which you will be introduced later, but a few of the more important are:

- **MultiRows and MultiColumns**  
These integer properties control the division of the DicomViewer into a number of images “cells”, a grid of the specified number of equally sized rows and columns, with the ability to display an image in each.
- **Images**  
This is a DicomImages collection (see below), an intrinsic and inseparable part of the DicomViewer, which holds the images to be displayed by the DicomViewer.
- **CurrentIndex**  
An integer index (1-based) specifying which of the images in the Images collection is to be displayed in the top-right cell.

#### 2.1.2 The DicomImages Collection

Like most objects, a DicomImages collection may be created as a “stand-alone” object, but a special collection exists as part of each DicomViewer. This object, accessed as the “Images” property of the DicomViewer, holds the images displayed by that DicomViewer. It has the standard methods and

properties common to all collections (see section 1.4 above), but the following additional method is used in this example:

- **ReadFile(string Filename) As DicomImage**

This is the main method used to read DICOM files from disk. The file is read, and added to the end of the DicomImages collection. The method also returns a reference to the added image, but this is not always used.

### 2.1.3 A DicomImage

This is the object used most commonly in DicomObjects, and encapsulates a DICOM image, whether that image has been read from disk (as above), generated “from scratch” or received over a network. It is important to realise that a DicomImage object is more than just the underlying DICOM data, as many of its properties, such zoom or rotation affect only how an image is displayed or otherwise handled within DicomObjects, without affecting the underlying data. The underlying data **can** be altered, but only through explicit use of the Attributes property or through a limited number of “named” properties. Some of the more commonly used methods and properties of a DicomImage are:

- **Width and Level (long integer)**

As is normal practice for radiological images, these properties allow the user to alter the mapping of the original DICOM data to the greyscale values displayed. They exemplify the principle stated above concerning underlying and transient information, as the initial values are derived from the suggested values in the DICOM data (if present), but changes to these values do not cause the DICOM attributes themselves to change.

- **StretchToFit (boolean) and Zoom (float)**

If StretchToFit is true, the image is scaled to show it at the largest size possible within its “cell” in the DicomViewer (and Zoom is ignored). Otherwise, the Zoom property controls the image’s magnification, a value of 1 causing one image pixel to correspond with 1 screen pixel. Many other properties exist, and will be considered later.

- **WriteFile(FileName as String, Part10 as Boolean, TransferSyntax as String, Quality as Variant)**

In a sense this is the reverse of the ReadFile method above, except that it applies to an image, not a collection, and it has more options, as it is necessary to specify features such as transfer syntax, which are determined automatically when a file is read.

Note: For most transfer syntaxes, the Quality parameter is ignored, but for lossy JPEG, it is interpreted as a quality factor from 0 (tiny but awful) to 100 (largest and high quality, but still lossy).

## 2.2 Writing an Image to Disk

Using the above objects, properties and methods it should not be difficult to see how to write the first DicomImage in the DicomViewer to disk in little-endian implicit VR format – the Visual Basic code being simply:

```
Viewer.Images(1).WriteFile "C:\Image1",true,"1.2.840.10008.1.2",0
```

Other languages will probably require a syntax closer to the following:

```
Viewer.GetImages().GetItem(1).WriteFile("C:\Image1",-1,  
↳"1.2.840.10008.1.2",Variant(0));
```

### 3 Simple Sending and Receipt of images over a network

Although storage of DICOM images in files is useful for testing and interchange on CD-ROMs etc, in the real world, (outside cardiology!), most DICOM images are transferred using the DICOM network protocols. Whilst the protocols are internally complex, and advanced programmers can access most of this complexity if they wish, DicomObjects provides several simple ways for you to do such transfers without worrying too much about the underlying transport system.

#### 3.1 Sending an Image

For this, the Send method of a DicomImage (or DicomDataSet) is required, and this takes just four parameters:

- IP address or other resolvable network name of the SCP
- Port number on which the SCP is listening
- The Application Entity Title (AET) of the SCP
- The AET you wish to call your application for this operation

Although some setting up of the SCP to receive images from your application may be required (most only accept from a list of “known” machines), this simple four parameter method is **all** you need to do to do a full transfer, as it will:

- Make an Association
- Negotiate presentation contexts (to include that for your image or dataset)
- Send the image using C-STORE
- Close the association
- Return the status it is sent (0 for success)

Because of this last point, the operation must be synchronous – i.e. it does not return until the transfer is complete (or fails). If any error occurs other than a non-zero status code (e.g. TCP connection failure), a trappable error will be thrown.

#### 3.2 Receiving Images

Every DicomViewer has the ability to listen on one or more TCP ports, and to receive images sent to it over the network. DicomServer objects have approximately the same functionality without the ability to display images, and unless noted explicitly, most of the properties, methods and events related to network activity apply equally well to a DicomServer.

As an utterly trivial example of how to receive an image (assuming you have an application or equipment that can send images), re-open the simple program created in the last chapter (just use points 1-3 if starting from scratch here), but then add and run code to perform the following method call:

```
Viewer.Listen 104
```

Next, set up your sending equipment or program to send images to your PC, using the following parameters:

- IP Address: That of Your PC
- Port : 104
- Called Application Entity: Anything (for now!)

Finally, try sending an image to the destination configured above – it should display on the viewer!

If you are conversant with the normal pitfalls and difficulties of sending DICOM images, the above may have seemed just “too easy”, but don’t worry, there are plenty of opportunities to apply those troublesome checks on IP address, application entity titles and allowed abstract syntaxes later on!

## 4 Query/Retrieve (SCU)

Retrieving information and images using the DICOM Query and Retrieve (Q/R) protocols is easy using the DicomQuery object in DicomObjects. By using the same object for both query (C-FIND) and retrieve (C-GET and C-MOVE) constant properties such as the application entry details of the server do not have to be re-entered.

There are three main methods, corresponding to the DICOM Q/R procedures:

- DoQuery (C-FIND)
- GetImages (C-GET)
- GetUsingMove (C-MOVE)

All these three methods are synchronous (i.e. they only return when network activity is complete), and they share most of the same properties to be used as part of their respective queries.

A few other methods also exist, which have some differences:

- **DoRawQuery**  
This is like DoQuery, but permits more flexibility, and is needed for worklist queries.
- **MoveSync**  
A synchronous move operation, only suitable for moving to other programs or machines.
- **MoveImages**  
This is an asynchronous operation, which return immediately.

Several properties and other features are common to all of the methods, so these will be covered first, followed by individual descriptions of the 6 methods.

### 4.1 Common Features

All these methods work by making an association to the remote machine (the SCP), including automatic negotiation of suitable presentation contexts, followed by the specific query/command required, and finally closing the association. All therefore need the following 4 properties to be set appropriately:

- Node: IP address or other resolvable network name of the SCP
- Port: Port number on which the SCP is listening
- CalledAE: The Application Entity Title (AET) of the SCP
- CallingAE: The AET you wish to call your application for this operation

DICOM supports many different “Roots” for the query/retrieve hierarchy, these being:

- PATIENT
- STUDY
- PATIENT/STUDY

You will need to check which of these is/are supported by your SCP, and set an appropriate value in the Root property of the DicomQuery. A 4<sup>th</sup> value WORKLIST is used for modality worklist queries, but these are covered separately (section 14 below) and will not be covered further in this section.

As it is common to make many consecutive calls to the same SCP using the same query root, these properties may usefully be set just once, and then re-used for subsequent queries.

The selection properties actually used to construct the queries (except for DoRawQuery) are:

- Name
- DateOfBirth
- Sex
- PatientID
- StudyUID
- SeriesUID
- InstanceUID
- StudyDate

#### 4.1.1 Date and Time ranges

Normally, DicomObjects attempts to coerce all values used to set DICOM attributes to the correct type according to the VR of the attribute involved, so time and date values passed as strings will be converted to date/time variants according to the computer's international settings, and then to DICOM format (YYYYMMDD etc.). However, DICOM allows ranges of dates for queries as:

- 20010101-20010103            1<sup>st</sup> to 3<sup>rd</sup> January 2001 inclusive
- 20010101-                    Any date from 1<sup>st</sup> January 2001 onwards
- -20010103                    Any date up to and including 3<sup>rd</sup> January 2001
- 120000-170000                Any time between 12:0 and 17:00

DicomObjects allows this type of value to be used for dates and times, but no conversions are applied, so it is the programmers responsibility to format ranges according to the DICOM requirements, as above, and DicomObjects does not itself enforce any formatting checks. The logic used when a string is assigned to a date/time attribute is:

- Attempt conversion to a single date/time
- If that fails, assume it is a range, and store unchanged

#### 4.2 DoQuery

This command sends a C-FIND query to the SCP, and returns the results, as a DicomDataSets collection, each member of the collection representing one response. The type of object represented by each response is set by the QueryLevel property which indicates the type of query being performed. Valid values are PATIENT, STUDY, SERIES or IMAGE. Of the selection properties, only those relevant to the selected QueryRoot and QueryLevel are actually used – e.g. for a DoQuery with QueryRoot=PATIENT and QueryLevel=STUDY, only PatientID, StudyDate and StudyUID are used, though StudyUID would normally be blank in this case.

DicomObjects requests a reasonably comprehensive list of return values, including all mandatory values. If any obscure values are required (and are supported by the SCP), then you may need to use DoRawQuery to add them as nulls to the query dataset.

#### 4.3 GetImages

Where the SCP supports C-GET (now increasingly rare!), this method allows simple synchronous image retrieval. All selection properties relevant to the QueryRoot are used, and the result of this method is a DicomImages collection.

One of the problems with C-GET is that the SCU must, during negotiation, pass a list of the abstract syntaxes (SOP classes) it is prepared to accept, yet until the query is run it does not necessarily know what the SCP will attempt to send it. DicomObjects copes with this by sending a list of all the

commonly used abstract syntaxes with every request, but if you need to, this list can be over-ridden using the TransferSyntaxes property or the AbstractSyntax registry entry (see section 17.3).

#### 4.4 *GetUsingMove*

As many SCPs now support only C-MOVE, this method provides a functional equivalent to GetImages, the main difference being that it uses C-MOVE, in place of C-GET. The internal mechanics are much more complicated, as it has to set up a listening port, send the request and receive the images on one or more separate associations, but the overall effect is a similar synchronous method. There are however, several important differences:

- The Destination property must be set to a value known to the SCP to represent this application. Note that this is the **only** information which the C-MOVE protocol allows the SCU to send regarding the image destination, therefore the SCP **must** be set up to map this name to the correct IP address and port number. Many users fail to understand that C-MOVE **cannot** be used without such configuration of the SCP, and that no DICOM SCU has any way of avoiding this necessity.
- The ListenPort property must be set to the value known to the SCP as above. This port must **not** be used for other purposes, and in particular, it should not be used in a DicomViewer.Listen or DicomSever.Listen call.

#### 4.5 *DoRawQuery*

Functionally, this is just like DoQuery, the only difference being that a user-supplied dataset is used to create the query, rather than using the selection properties to create one. This is slightly more work for the programmer, but does then allow total flexibility, as any of the query and return values may be included. Please note that the QueryLevel property is in fact a member of this dataset, and an appropriate value for this attribute (0x0008,0x0052) needs to be included.

#### 4.6 *MoveSync*

This method simply issues a C-MOVE request, instructing images to be moved to the AET given by the Destination property. It waits for the operation to complete (by which time any secondary C-STORE operations must also be complete), then returns the final status of the C-MOVE operation. This method is ideal for moving images to a different application (whether on the same machine or another), but it **must not** be used to move images back to the same application. The reason is that such images could not be accepted until the AssociationRequest and ImageReceived events had fired, but these cannot fire while this (synchronous) operation is waiting for completion, so a deadlock would occur. For moves back to the originating application, use GetUsingMove, MoveImages, or DicomConnection.Move instead.

#### 4.7 *MoveImages*

This method is like MoveSync above, but it after making the initial association, it returns immediately. The result is that it can be used to move images back to the originating application, avoiding the deadlock described above, but it has a major problem in that no success or error indications can be returned. This method is therefore not recommended for new applications, and one of the other Move methods should be considered.

#### 4.8 *DicomConnection Based Q/R Methods*

With the exception of MoveImages (which has no error return), all the DicomQuery-based methods are synchronous, and therefore tie up the whole application while they are completing. There are some occasions, however, where it is desirable to run these methods in the background, and this can be done

using an asynchronous DicomConnection object, using its Find, Get and Move methods. All these are like DoRawQuery, in that they require you to provide a complete valid query dataset, so they are slightly more work than the simpler DicomQuery-based methods, but they do provide total flexibility, asynchronous operation, and full error reporting. They are therefore the recommended query methods for advanced users.

Please note that the SOP classes required for these methods are **not** included in the default list offered by an outgoing DicomConnection. Therefore, if you intend to use these methods, you must specify them explicitly before calling SetDestination – e.g.

```
Set connection=viewer.New("DicomConnection")
connection.Contexts.Add "1.2.840.10008.5.1.4.1.2.1.1" `Patient Root FIND
connection.Contexts.Add "1.2.840.10008.5.1.4.1.2.1.2" `Patient Root MOVE
connection.Contexts.Add "1.2.840.10008.1.1" `Verification
```

Note that if you do this, **all** required SOP classes must be added as above, as the default list is not used if there are any manual additions.

## 5 Off-line Media

Although DICOM files are used regularly on their own, especially by developers, such image files (as opposed to network operations) are only recognised by the DICOM standard as part of a “Fileset” on defined media, using approved filenames, and accompanied by a DICOMDIR directory file. Many of these requirements such as the requirement to use ISO 9660 format for CDs are beyond the scope of DicomObjects, but it does provide support for reading, writing and modifying DICOMDIR files, as well as the actual images.

The structure of a DICOMDIR file is actually quite complex. Logically, it is arranged hierarchically, the normal 5 layer hierarchy being:

- Fileset, which contains one or more
- Patients, which have one or more
- Studies, which have one or more
- Series, which have one or more
- Images, which reference the actual image files, and may contain icons  
(Other structures are possible but rare)

However, the data items are arranged as a single sequence within the DICOMDIR file, using a complex series of file offsets to define the hierarchical structure. It would clearly be very difficult for a high-level programmer to generate these offsets, so DicomObjects presents the DICOMDIR to the programmer using the hierarchical structure, and does any necessary offset calculations only during reading and writing. A DICOMDIR is read as a DicomDataSet, representing the Fileset object, and the next level of the hierarchy (normally representing individual patients) can be found as the items in its Children collection, and these in turn have Children properties containing the studies, and so on down to images. This same structure may be modified or built from scratch to allow modification and writing of DICOMDIR files.

### 5.1 Reading

Reading a DICOMDIR file is very easy – just create a new DicomDataSets object, use the ReadDirectory method, and the whole hierarchical structure referred to above will be found in the resulting DicomDataSet.

To extract all the information within it, just iterate through the members of the datasets Children collection, each of which should represent a patient. Each of these will have a Children collection representing studies, and so on down to images. Note that you should, at each level, check the directory record type attribute (0x0004,0x1430), to ensure that it is as expected (PATIENT, STUDY etc.), as some manufacturers (quite legally) add PRIVATE items, which can cause confusion if not skipped.

#### 5.1.1 Finding the Image File

Once the lowest level of the hierarchy is reached, the dataset should represent a single image, but you then may need to find the filename of the image file it references. The required attribute is 0x0004,0x1500, and the value of this is an array of strings, representing the path from the DICOMDIR, to the image file. e.g. if the directory is at D:\DICOMDIR, and this attribute returns a 3 value array containing “IMAGES”, “PAT02”, “IMG0004”, then the image file can be found at:

D:\IMAGES\PAT02\IMG0004

Sample VB code is as follows, but other this is slightly more complex in other languages due to the need to interpret SAFEARRAYs correctly.

```
Path=Left(dicomdirpath, Len(dicomdirpath)-9)
```

```

PathArray=imagedataset.Attributes(&h0004,&h1500).Value
For i=LBound(PathArray) to UBound(PathArray)
    Path=Path & "\" & PathArray(i)
Next

```

### 5.1.2 Extracting Icons

Some DICOMDIR files contains icons representing the referenced images, enabling the user to be shown low resolution images, without having to read and decimate every large file in the FileSet. If present, this appears as a sequence attribute (0x0088,0x0200), normally in an IMAGE level dataset, but it may sometimes be found in a SERIES dataset. The icon itself is like a minimalist image, with pixel data, and immediately associated attributes such as size and format, but no demographic data. Due to the way that DicomObjects is structured, extracting this item (the one and only item of the sequence) will result in a DicomDataSet object **not** a DicomImage, but fortunately, this is not a problem, as a DicomDataSet may be added to a DicomImages collection, and the result is (perhaps surprisingly) a DicomImage.

The following Visual Basic code shows how to add the icon from an image dataset to a DicomViewer (viewer):

```

Dim iconattribute as DicomAttribute
Dim iconSequence as DicomDataSets
Set iconattribute=imagedataset.Attributes(&h0088,&h0200)
If iconattribute.Exists Then
    Set iconsequence=iconattribute.Value
    Viewer.Images.Add iconsequence(1)
End If

```

## 5.2 Creating

It is perfectly possible to build a DICOMDIR structure in a new DicomDataSet, and then add items hierarchically through the Children collections (and indeed this was the only method prior to version 4), but there is now a much easier method, called AddToDirectory. This takes an image and a few other details as its parameters, and adds all necessary elements to the dataset, ensuring not only that every patient, study etc. has entries, but also arranging the correct relationships, and avoiding duplicates. In addition, icon images may easily be added to the data. This massively reduces the amount of work involved in making a DICOMDIR, but there are still some things you need to consider:

- You need to choose your own file and path names, and ensure these are not duplicated within a FileSet
- You must ensure that file and path names correspond to the requirements for your chosen media. For CDs, this means that they must be 8 or less characters, upper case, and with no extensions.
- Note that the filename you pass to the AddToDirectory method must be relative to where you intend to write the DICOMDIR.
- When calling AddToDirectory the first time on a new dataset, basic “top level” information is filled in, including a FileSetID (0x0004,0x1130), but you may wish to check and amend this. You should, where possible, also arrange for the FileSetID to be written as the volume label when burning onto a CD or writing to MOD etc.
- Some profiles require attributes beyond the basic ones added by DicomObjects. To help you to add these, the return values from AddToDirectory is itself a DicomDataSets collection, with four items, referencing the patient, study, series and image datasets respectively. If required, further attributes may be added to any of these.

The following example demonstrates many of these features, adding all the images in collection “images” to a new dataset, assuming that it is to conform to the STD-XABC-CD profile, which requires data of birth, sex, and performing physician (and a few other values not shown) in addition to the “normal” requirements.

```
Dim directory As New DicomDataSet
Dim levels As DicomDataSets
Dim Image As DicomImage

rootpath = "D:\\"
TransferSyntax = "1.2.840.10008.1.2.1"
For i = 1 To Images.count
    Set Image = Images(i)
    Set levels = directory.AddToDirectory(Image,path,TransferSyntax, 128)
    ` PATIENT LEVEL - Date of Birth
    levels(1).Attributes.Add &H10,&H30,Image.Attributes(&H10,&H30).Value
    ` PATIENT LEVEL - Sex
    levels(1).Attributes.Add &H10,&H40,Image.Attributes(&H10,&H40).Value
    ` SERIES LEVEL - Performing physician
    levels(3).Attributes.Add &H8,&H1050,Image.Attributes(&H8,&H1050).Value
    ` IMAGE LEVEL - image type
    levels(4).Attributes.Add &H8,&H8,Image.Attributes(&H8,&H8).Value
    Image.WriteFile rootpath & path, True, TransferSyntax, ""
Next

directory.WriteDirectory rootpath & "DICOMDIR"
```

### 5.3 Updating

To update a DICOMDIR, simply read using ReadDirectory, modify as necessary using the Children hierarchy, and write out again using WriteDirectory. Any necessary changes to the offset values will happen automatically.

### 5.4 Multiply Referenced Directory Records

DICOM allows a variation on the neat hierarchical structure above whereby a record can be referenced by multiple other records, a “Multiply referenced directory record”, or MRDR. DicomObjects supports such records, through the MRDR property, which is documented fully in the help file, but if such records are used, it is your responsibility to maintain the directory structure, using the MRDR property. This type of record is not used by the AddToDirectory method, but DicomObjects does still handle offset calculations during both reading and writing.

## 6 Printing

Printing to a DICOM printer and printing to a Windows printer are completely different operations, but both are possible using DicomObjects. Due to the multiple possible layout variations, native DICOM printing is a complex procedure, and in previous versions of DicomObjects, all the separate normalised operations required needed to be invoked independently. Fortunately however, version 4 has made this whole process a great deal simpler, using the new DicomPrint object. If you wish to have total fine-grained control over printing, the original normalised operations do of course remain available. Printing to a Windows printer tends to be quite different from different development environments, but some examples are presented after the sections on DICOM printing.

### 6.1 Printing Using DicomPrint

#### 6.1.1 Minimal Requirements

If you have a set of DicomImage objects in a collection called *images*, and if the DICOM printer is set up to recognise your computer's IP address as a valid source, then the following code should be all that is required to print images on to film, using a 2x3 format:

```
Dim printer As New DicomPrint
Dim Image As DicomImage

printer.Node = RemoteIPAddress
printer.Port = PrinterPort
printer.CalledAE = PrinterAET
printer.CallingAE = MyAET

printer.Open

printer.Format = "STANDARD\2,3"
printer.Orientation = "PORTRAIT"
printer.FilmSize = "14INx17IN"

For Each Image In Images
    printer.PrintImage Image, False, True
Next

printer.Close
```

Of course, the **bold** items will need to be modified to those appropriate for your printer, and may need to be obtained from the service engineer. You may also need to check and modify the format, orientation and filmsize according to your printer's capabilities.

Whilst this code will print basic images, many enhancements are possible within DICOM printing, and are supported by DicomObjects. The following sections explain some of these.

#### 6.1.2 Annotating Images using DicomLabel objects

One of the fundamentals of DICOM printing is that images are sent to the printer **exactly** as they are to be printed, purely as pixel data, with no subsequent modifications at the printer, so any labelling such as patient demographics needs to be added before the image are sent. In DicomObjects this is done by attaching annotations to the images' Labels collections, exactly as can be done for display purposes. e.g. to show the patient's ID and the study date at the top left hand corner of an image, the following code could be used:

```
Set l=image.Labels.AddNew
l.LabelType=doLabelText
l.Text=image.PatientID & " " & image.Attributes(0x0008,0x0020)
l.Top=10
l.Left=10
```

When printing the image, the final parameter to the PrintImage method must be true.

For more details of adding DicomLabels, see section 8.3 below.

### 6.1.3 DICOM “Annotations”

“Annotations” as applied to DICOM printing is a much misunderstood term, as it applies to one or more text strings printed once each on the film as a whole, rather than the more commonly used demographic or graphic information printed over individual images. In most cases, this facility is used only to print an institution name, but can be used to add a patient name etc., if the space allowed is long enough. Unfortunately, annotation formats vary widely between printers, and there may be many different possibilities, so it is important to check the printer’s conformance statement, and locate the following information:

- The name of the format you wish to use
- The positions, numbers and permitted lengths of the annotations permitted by that format.

Once you have done this, you can add the following into the basic code above, at any stage before the first PrintImage command:

```
Printer.AnnotationFormat=format
Printer.AddAnnotations annotationID,text
```

If a format supports multiple annotations, then the last line may be repeated as necessary.

In general you should be very careful when using printer annotations, as they can easily introduce avoidable printer-specific dependencies.

### 6.1.4 Preformatted images

If you wish to have total control over the printing process, you may create your own preformatted images for printing, and send them directly to the printer by setting the Raw parameter of PrintImage to true. This may be useful where you wish to apply custom overlays etc. on the data. If however you do this, no windowing or other transformations will be performed by PrintImage, so this must have been done previously, normally by using the DicomImage.PrinterImage method (as is used internally by PrintImage when Raw is false).

### 6.1.5 Checking Printer Status

It is always a good idea before starting printing to check whether the printer is actually available, to make sure for instance that it is not out of film. One way of doing this is to use the Printer property of the DicomPrint object immediately after the Open command. In particular, the printer status attribute (0x2110,0x0010) of this dataset should be checked, as it should be NORMAL.

Please note that this property is retrieved only as part of “Open”, and is not automatically updated. Should an update be needed in the middle of printing, see the following section on other normalised operations.

### 6.1.6 Applying other normalised operations

It is possible using DicomObjects to do DICOM printing using a DicomConnection object without using a DicomPrint object at all (as described in the next section). This gives you complete flexibility to use whatever normalised operations you want, but it is a lot of work, as you would duplicate all the operations of DicomPrint. Instead, where you need to do something slightly unusual, you can use the

DicomPrint's Connection property (anywhere between Open and Close), to achieve similar effects. For instance, to retrieve an updated equivalent of the Printer property you could use.

```
Set nullDataSet=new DicomDataSet
print.Connection.NGet, doSOP_Printer,doInstancePrinter
Set newPrinter=print.Connection.ReturnedDataSet
```

Note that the DicomConnection method used by a DicomPrint object operates synchronously.

## **6.2 Printing Using Normalised Operations**

This was the only method of printing in version of DicomObjects prior to version 4, and is considerably more work than using the DicomPrint object, but it does have a few advantages over the newer simplified method, which may be important to some users:

- You have complete control over all operations
- Print data can be sent asynchronously if you wish
- N-EVENT-REPORT notifications can be received, and under some circumstances (if the DicomConnection object is run in mode doNoSync), a response can be sent.

## **6.3 Printing DICOM Images to a Windows Printer**

Whilst printing to a DICOM printer is important, many users need to be able to print to other printers through Windows. This is one area where different development environments have different mechanisms available, but most will in some way support the Windows "IPicture" interface, so although the code that follows is specific to Visual Basic, similar procedures should apply in other languages, and will be added to later version of this guide.

The main property used for this purpose is the image's Picture property, which returns an object containing a bitmap representation of the image in a language neutral format. In most environments, this can be used directly either to send to a printer, or to fill an object in a form or print template. e.g. in Visual Basic, the following code is all that is required to print an image to the currently selected default printer:

```
Printer.PaintPicture image.Picture
Printer.EndDoc
```

Of course, for accurate placement on a page, and scaling, more arguments to PaintPicture would normally be given.

## 7 Exporting DICOM Images to Other Formats

For use in applications which do not support DICOM, export to various standard formats is supported. The formats with native support are JPEG and Windows BMP, though other formats may be supported using custom DLLs, and an experimental TIFF import/export DLL is available from Medical Connections if required. For multi-frame images, export to AVI files is possible.

### 7.1 Single Frames

The 3 appropriate methods for single frames are:

- FileExport
- MemoryExport
- ArrayExport

All these three use the same routines internally, but FileExport writes to a file, MemoryExport puts the data in a global memory block (and returns the handle to it), and ArrayExport puts the data in a SAFEARRAY. The help file provides further details of the parameters required for each method.

As the image produced is only 8 bit, normal windowing and lookups (as used for display) are applied to the data, though the final screen linearisation (CalibrationCurve) is not applied, as images should be portable.

Where these methods are used on a multi-frame image, just the currently selected frame, as selected by the Frame property of the image is used.

### 7.2 Multi-frame Images/Cine

WriteAVI is a new method in version 4 of DicomObjects which allows all or part of a multi-frame image to be exported in a standard non-DICOM manner. This method is very flexible, allowing choice of both CODEC and quality either at design or run-time. The parameters are described in the help file, but some further notes may be useful:

- In order to enable run-time choice of compression details, set the ShowDialog parameter to true.
- If you wish to suggest initial values for the dialog, this can be done by setting the CompressorCode and Quality values in the call, though these can be over-ridden by the user if ShowDialog is true.
- For development use, it is useful to use the value SHOW for the codec, as this provides a method of seeing the four character code for the codec chosen.

### 7.3 Non-file Exporting

Where it is necessary to export image to destinations other than files, they may often be exported using the Copy (to clipboard), or Picture methods, as both produce standardised representations of the image, as displayed.

## 8 Advanced Image Review station

One of the main uses for DicomObjects is the production of image review stations, and many of the toolkit's features are designed around this need. Before displaying images, it is necessary to locate the images required, then receive them from a network or read them from a disk, and these "filing" topics have already been covered elsewhere in this manual, so this section concentrates specifically on how best to present images to a user.

Of course, this is the type of application where user interface design is of paramount importance, so although the Visual Basic example can get you started, the details of which buttons do what is entirely up to you – that is why DicomObjects itself has no intrinsic user interface.

### 8.1 Basic Viewing Controls

#### 8.1.1 DicomViewer properties

A DicomViewer displays any number of images in a matrix defined by its MultiRows and MultiColumns properties. The images available for display are held in its Images collection, and the index of the top left image is CurrentIndex, which may be modified to control which images are shown. If necessary, the images may be edited using the DicomImages.Move or Remove methods. In addition, it is possible to have more than one copy of an image in the collection – each will have its own display parameters.

By using the above properties, the user is able easily to control the format of the display, and to select which images to display. When designing the user interface for this functionality, it is often useful to highlight images using their BorderColour and BorderWidth properties, and it is easy to tell in which image a user has clicked the mouse by using the DicomViewer's ImageIndex method. See the help file for more details of these.

#### 8.1.2 DicomImage properties

Each image has a set of display properties, which are simple and almost self-explanatory.

- Zoom (not used if StretchToFit is true)
- OriginX and OriginY (control panning)
- FlipState and RotateState
- Width and Level (control windowing – i.e. brightness and contrast)
- StretchToFit

These are present in just about every image browser, and users will expect quick and easy access in some way to them all. DicomObjects does however provide much more than this, and the more advanced facilities will now be discussed.

### 8.2 Multi-frame (Cine) Images

#### 8.2.1 Self-running Cine

In previous versions of DicomObjects, the only way of choosing which frame to display was to change the Frame property of an image. This works well, and gives the programmer considerable flexibility, but users increasingly expect real-time cine display, especially of cardiology images, and version 4 now allows such images to "run" autonomously. Only two properties are needed to control this:

- CineMode
- CineRate

By setting CineMode to an appropriate value, the display of an image will run either once or continuously forwards or backwards through the available frames, or may even oscillate (forward and back). The rate at which this happens is controlled by CineRate, a floating point value, which is multiplied by the rate specified within the file itself to give the actual display speed. If you need to make display changes as the cine runs, such as a frame counter or progress bar, use the OnFrameChanging event.

### 8.2.2 Masking

Another feature commonly used in vascular/cardiological imaging is subtraction, where one frame of an image (the “mask”) is subtracted from all other frames before they are displayed, the aim being to show contrast, and eliminate the surrounding anatomy. To use this feature, simply set the image’s Mask property to the frame number required. Note that masking may be specified in the image data itself, in which case Mask property will be set to the specified initial value. However, if this is frame 1, then the image initially presented to the user may be blank, so you may wish to check this initial value, and set Frame to an appropriate alternative value. More complex masking, using multiple averaged frames for the mask, and mask shifting is supported, but due to the complexities, this can only be specified using a Presentation State (see section 8.4 below for more details).

## 8.3 Annotations

After the image itself, the next feature that most users wish to see within their viewers is annotations, provided in DicomObjects by DicomLabel objects. They have many uses, but the main ones are:

- Provide demographic and study related information
- Highlight areas of interest to other users
- Mark regions of interest for measurements

DicomLabel objects can be used for all the above purposes, but before going further, it is worth reviewing their main properties.

### 8.3.1 Types of DicomLabel

There are 10 types of label (the last 3 of which are new to version 4.1):

- Text
- Ellipse or Circle
- Rectangle or Square
- Line
- PolyLine
- Polygon
- Bitmap
- Arc
- Special
- Interpolated

The type of label is selected using the LabelType property, and according to this, other properties of the label may or may not be used – e.g. LineWidth is not relevant to a normal text label.

However, labels are also categorised in another way –according to their relationship to their viewer or image, and three possibilities exist:

- **Attached to a DicomViewer’s Labels collection**

In this case, the label is displayed at “full” size, with all measurements in screen pixels, relative

to the top left corner of the DicomViewer. The number and content of any images held by the viewer does not in any way affect the label.

This type of label is commonly used to add demographics known to be common to all images. It is also used during drawing operations – see below for details.

- **Attached to a DicomImage’s Labels collection, with ImageTied=false**

In this case, the label is displayed at “full” size, with all measurements in screen pixels, but they are relative to the top left corner of the area of the viewer within which the parent image is displayed, and the label is clipped to that area. The label is only displayed if the associated image is shown, but it is unaffected by the zoom, rotation, etc. of the image.

This is the type of label normally used for demographic and image information, as it can remain visible as long as the image is.

- **Attached to a DicomImage’s Labels collection, with ImageTied=true**

In this case, the label is zoomed, rotated scrolled etc. with the image, with all measurements in image pixels, relative to the top left corner of the area of the image, but clipped to the area within which the parent image is displayed.

This type of label is used wherever it relates directly to the underlying pixel data, and is the only type which can be used for region of interest (ROI) calculations.

### 8.3.2 Adding Demographic Labels

In general, demographic labels are easy to add – just extract the required attributes from the image, create a DicomLabel, set its colour, position, type etc, and add to the appropriate Labels collection. There are however a few points worth considering:

- Where there is operating system support (mainly on Windows 2000 and above) consider extracting names etc. using the UnicodeValue property instead of the default Value property. The DicomLabel’s Text property accepts Unicode values, and this procedure will allow use of localised names.
- If left as 0, the width and height of text boxes are assumed to reach to the bottom and right of the clipping region for that label (whether defined by image cell or viewer size). Whilst this works for simple labels, it can cause unexpected effects when alignment is anything other than doAlignLeft, or when the text is rotated (Angle != 0), and so explicit sizes are highly recommended.

### 8.3.3 Adding Orientation Markers

A new facility in version 4.1 is the ability to specify the “edge” of an image to be labelled, and to leave DicomObjects to determine the corresponding anatomical orientation, using the information within the DICOM dataset – this is done using “Special labels”. This is a powerful new tool, and full instructions can be found in the help file under “Special” labels, but it is important to note that errors in this area could have serious clinical consequences, so the following guidelines are important:

- Always ensure that sensible, small values are used for height and width, so that the position remains correct when the image is flipped or rotated
- Always use markers in pairs, as this ensures that they are interpreted as orientation markers, rather than “side markers” – e.g. an “L” on it’s own adjacent to a leg could be misinterpreted as indicating the left leg, rather than perhaps the left side of the right leg.
- Where the language used is not English, the characters used can be changed using the DirectionStrings property of a DicomGlobal object.

### 8.3.4 Adding Graphical Annotations

As graphical labels are normally related to features in the underlying pixel data, they are normally drawn freehand by the user, using the mouse or similar. As a result, there is a need to be able to:

- Draw them in a reversible manner
- Attach to the correct image
- Ensure that the coordinate system used for drawing is compatible with the system used for storage

DicomObjects provides methods to achieve all these needs. They are included in the visual basic demonstration, and can be shown in action by selecting the “Annotation” button for the mouse action. The code is mainly in the MouseDown/Move/Up events, and can at first appear a little obscure, especially as there are minor differences for different types of label, so here is a summary of the steps used.

- In MouseDown, note the initial coordinates, and use ImageIndex to determine which image was clicked in.
- Create a suitable DicomLabel object, but do not attach it to either the Viewer or the Image’s Labels collection.
- Instead, set its XOR property to true (to enable reversible drawing), and draw directly onto the viewer using the DrawLabel method.
- In any calls to MouseMove, remove the previously drawn label simply by drawing it again (in XOR mode), update it as necessary (add points or change size depending on type), and then redraw. This will produce good visual feedback for the user.
- In MouseUp, remove the final temporary rendition of the label by drawing it again. If attaching the final label to the viewer (uncommon), then the final label may simply be added to the viewer’s Labels collection, but if it is being added to an image (as determined in the first step), then it must be rescaled to match the image’s coordinate system, before being added to the image’s Labels collection. To do this, set the label’s ImageTied property as required, then call its Rescale method, with the target image as the parameter.

Of course, this is an extremely simplistic example, using only one mouse button, and improvements are easily made.

### 8.3.5 Using Annotations as Regions of Interest

One of the new features added to version 4 at user request is the ability to use DicomLabels directly as regions of interest. Having drawn an area as above, this allows a user easily to be shown various statistical measures of the region. Note that these are only available where the label is a member of an image’s Labels collection, and its ImageTied property is true, as this enables direct correlation with the underlying pixel data. Attempts to use any these on other label types will generate an error.

The statistical properties come into two broad categories:

- The properties ROIArea and ROILength calculate the area and line length of the label, and then use the scaling attributes of the image (where present) to translate these into “real world units”. The type of units applicable can be obtained from theROIDistanceUnits property, which may be “mm”, “mm at Imager”, “mm at Imaging Plane”, or if no scaling information is present in the image, “Pixels”. For ROIArea, these should be course be regarded as square units.
- The second group of properties relates to the underlying pixel values. The most basic is ROIValues, which returns the values themselves, but ROIMean, ROIMin, ROIMax, and ROIStandardDeviation are derived from its values. These values take into account any modality transformations, whether this be a rescale and intercept (as used for CT images), or a

modality lookup table. For CT the result should be in Hounsfield units, but for most other modalities, the result is in arbitrary units.

### 8.3.6 Hit Testing

If you wish to offer users the chance to delete, modify or reposition labels after they have been drawn, it is important to be able to check for clicks onto them. For this purpose, the LabelHits method returns a list of suitable candidates – see the help file for more details.

### 8.3.7 Saving Labels

At present, there is no standardised method of saving DicomLabels, as they are unique to DicomObjects, and would not make sense in other environments. In the next version of DicomObjects there will be the facility to convert all display information, including any attached labels into a DICOM presentation state, but until that facility is available, there are a few alternatives:

- Make a sequence of private attributes, holding all the properties relevant to your chosen labels, and add that to the underlying image. Note that as of version 4, all properties of a DicomLabel, including Points, are read/write.
- Save the information in an external file.
- Make a copy of the image with the labels “burnt-in” to the pixel data, using the Capture method.

## 8.4 Lookup Tables

In versions of DicomObjects prior to version 4, the only transformations applied to the pixel data before display were:

- Rescale intercept and slope if appropriate
- Window Width and Level
- Reversal if photochromic interpretation=MONOCHROME1

However, in version 4 and above, the full DICOM “pipeline” is implemented, which allows several lookup tables (LUTs), and consists of:

- Rescale values **or** Modality LUT
- Window and Level **or** Values of Interest (VOI) LUT
- Presentation LUT
- Display Linearisation LUT

All these except the last are contained within the image data, and their use is controlled by the image’s ModalityLUT, VOILUT and PresentationLUT properties, which all default to 1, meaning that if present, the first LUT in the data shall be used. If more than one LUT is present, any one may be selected by setting the appropriate property to the correct index, or, if wished, the properties may be set to 0 to disable a particular table. Note that a VOI LUT, if present, over-rides the Window and Level properties, and this does sometimes cause confusion if the relationship is not understood, especially as it is a change from the version 3 behaviour.

The final linearisation LUT is provided to comply with part 14 of DICOM, the “Grayscale Standard Display Function”. Full details are provided in the help file under the controlling property, which is CalibrationCurve.

## 8.5 DICOM Greyscale Presentation State

One of the important recent additions to the DICOM standard is the “Grayscale Softcopy Presentation State” objects, referred to from here on just a presentation state. This is a DICOM dataset which can be

independently stored and transmitted, and should be included as part of a study, but instead of containing image data, it defines how other images should be displayed. DicomObjects provides direct support for displaying images using these objects, by setting such a dataset as the PresentationState property of an image. When this is done, it over-rides **all** other display parameters such as windowing, zoom, flips etc., as this is the intention of such a dataset. The only external lookup table still used is the calibration curve, as described in 8.4 above. The contents of presentation states are well documented in the standard, and they are relatively easy to construct from scratch, but in a viewer, you are for now more likely to be using those created elsewhere, so here are a few aspects to consider:

- If you wish to retrieve these using C-GET, you will need to add the appropriate SOP class UID (1.2.840.10008.5.1.4.1.1.11.1) to the list of classes in the AbstractSyntax registry key (see section 17.2).
- If retrieved via C-GET or C-MOVE, the presentation state will appear as a DicomImage, not a DicomDataSet. This does not matter, provided you do not attempt to display the item itself, as assignment to another image's PresentationState property will still work.
- There may be more than one possible presentation state for a given image – if so you should offer the user a choice of which to apply, using any fields within them suitable for identification.

This is a relatively new feature of DicomObjects, which very few developers have yet utilized. Whilst it has been tested against all known test images, real-world experience is limited, and feedback on any issues discovered would be appreciated.

## **8.6 Display Speed Optimisation**

Display speed of DicomObjects is now very much faster than in version 3, as DirectDraw is used throughout where available, but there are times when there is a trade-off between display speed and RAM usage, so this section explains the properties and variables available to help achieve the correct balance.

### **8.6.1 Compressed Image Caching**

The normal policy adopted by DicomObjects for the handling of compressed images is to decompress frames only when necessary, and to keep the decompressed versions only for as long as is necessary for display or translation, thereby minimising image load times, and memory use. This behaviour is not, however, always appropriate, especially when cine sequences are shown. All frames can be decompressed at once by using DecompressAll, and a particular frame can be decompressed using DecompressFrame, or, and this is often the best option, the CacheDecompressed option can be set true. When this property is true, then frames are still decompressed only when necessary, but the decompressed copy is then kept either until the image is destroyed, or CacheDecompressed is set false.

### **8.6.2 Display Caching**

Normally DicomObjects keeps a copy of each frame as it displays it, clearing this cache only when display parameters change. This is useful when displaying multi-frame images such as cardiac angiography, as it can increase the speed dramatically, but if runs are long, then the memory used can be large, and memory paging may have a paradoxical negative effect. If so, then such caching can be disabled by setting the CacheDisplay property to false.

### **8.6.3 DirectDraw**

Normally, DirectDraw is used wherever possible, but it can be disabled if necessary by setting the DirectDraw registry value to 0. You can also check if DirectDraw is being used by using the

DicomViewer's `isUsingDirectDraw` property. If this is unexpectedly false, then setting the `DirectDraw` registry value to 2, will cause a diagnostic message box to display when the first image is shown.

#### **8.6.4 System Settings**

Often, the largest improvements in speed can be achieved by upgrading to newer display drivers, and other system settings can also be relevant e.g.

- On Windows 2000, there is a feature, enabled by default, which adds a “shadow” behind the mouse cursor. Some display systems cope very badly with this, and on such systems, disabling the shadow (through mouse properties in control panel) can lead to a dramatic increase in speed.

## 9 Web Usage

In general, most applications that can be run as stand-alone programs can now be run in alternative forms on either web clients or on servers, so the only real constraint is your ingenuity, but there are a few specific issues which need to be considered when using DicomObjects in these slightly unusual environments.

There are many ways of handling DICOM in a web environment, and this chapter explores the two extreme approaches, one in which DicomObjects is run entirely on the web server, passing nothing but html and jpeg images to the client – this method requires a Windows server, but will run with any client. At the other extreme, all DICOM functionality may exist on the client, which downloads script from the server, but then communicates directly with a DICOM SCP. In this case, the client must run Windows, but any web server, and any DICOM SCP can be used. Both the examples consider the simple process of locating, retrieving and display/manipulating images, as this is the commonest type of electronic patient record type task required of web applications in a hospital environment, but in principle, almost any other functionality covered in this manual could be achieved in a web environment. Of course, hybrid solutions are also possible, running DicomObjects on both the server and the client.

It is worth noting that passing everything through a web server may have some costs in terms of speed, but a major gain is security. As currently implemented (prior to the September 2001 security enhancements which will not be in common use for many years), DICOM has virtually no means of user authentication other than IP address and AET. Whilst web authentication does have some weaknesses, it is massively better than anything currently available in DICOM itself, so image access through a web server can be much better controlled and identifiably logged than native DICOM queries from a workstation.

### 9.1 Running DicomObjects on a Web Server

Of course, in order to use DicomObjects on your web server, the server must be running a version of Windows, and the web server must support a scripting language. In most cases, this means running Microsoft IIS, but other Windows servers do exist. On this basis, the examples in this section will be shown in VBScript, as used in Microsoft Active Server Pages, but the principles would apply to other scripts. For information on usage in VBScript in general, see section 15.2 below.

#### 9.1.1 Issuing Queries

For an experienced web application developer taking the results from a web form, and turning them into a DICOM query using a DicomQuery object should be fairly trivial, and the web example included with the download should show you where to get started. Likewise, iterating through the results, and turning them into valid, hyperlinked html is not difficult, once you know which attributes of the result to use. One problem you may find is that many SCPs return only a small number of fields to C-FIND queries, so important information such as a study description may be missing. The download example gets around this by retrieving a full image and extracting the information from that, but you need to think about your preferred solution to this problem as:

- If the web and DICOM servers are separate machines, this could be a lot of wasted transfer – imagine if the image is a 100 frame cardiac cine!
- C-GET is not now widely supported, and the equivalent using C-MOVE is slightly more work – see below for details.

#### 9.1.2 Retrieving Images

There are 3 reasons why you may wish to retrieve images directly into your web server:

- To convert to other formats before passing to the client
- To extract information from the image
- To push on again using DICOM to the client, thereby improving security and allowing transfer syntax conversions.

If the SCP supports C-GET, then the GetImages method is perfect for this type of application, as it is synchronous – you send the request, wait for the result(s), and then use them as you need. However, most servers now support only C-MOVE, and this is problematic for scripting applications such as Active Server Pages, as you have no method of setting up an independent object such as a DicomViewer or DicomServer, then acting on its events when the images arrive. Fortunately, version 4 of DicomObjects does provide an answer – GetUsingMove, which was designed specifically for this type of application. To the server, it appears as a normal C-MOVE, but to you, it has most of the simplicity and synchronicity of C-GET. Fuller details of this method are provided in the help file, and in section 4.4, but I will stress one important fact again, as it causes so many problems.

To use GetUsingMove (or any other C-MOVE operation), your SCP must be set up to “know” your AET, and how to translate it into an IP address and port number - there is no alternative!

### 9.1.3 “ON the Fly” DICOM Conversion

If you wish to send images to any basic browser, without worrying about its type or security settings, then there is no better method than sending a JPEG image. As this was one of the first uses of DicomObjects, there is in fact a specific DicomImage method for this called “JPEG”, but this method is now deprecated, and the ArrayExport method is preferred, as it is more flexible. Assuming that a required image has been retrieved or read into object *image*, then the only code required to send it as a jpeg image of quality Q is:

```
result=image.ArrayExport("JPG",Q)
Response.ContentType = "image/JPEG"
Response.BinaryWrite result
```

The same could be used to send a BMP file if you do not wish to have any degradation. For a cine image you could use WriteAVI to a temporary file, followed by a redirect to that file.

Note that images produced this way appear (apart from any compression) just as they would be displayed, after windowing etc., as only 8 bit data can be included in “standard” file formats. You may therefore need to provide a facility for your users to change the Window and Level properties before the conversion is performed (as shown in the example).

### 9.1.4 Threading/Responsiveness Issues

IIS can be run in many modes, with different numbers of threads, and pools of connections etc. For this type of application, image retrieval and sending may take significant amounts of time to execute, so it is important to ensure that sufficient threads are available to prevent one request blocking other requests.

### 9.1.5 Licensing

Unfortunately, there is no way to “embed” the licensing information held in DicomObjects.lic into a web script, which therefore must always effectively run in “design” mode. As a result, the conventional method of licensing control does not work, as any user with access to the server has access to the license file, which could be mis-used. Therefore, for this type of application, a special version of DicomObjects has been made, which relies instead on other mechanisms – please ask Medical Connections for details.

## 9.2 Running DicomObjects on the web client

As an alternative to putting the code on the server, you can have the server simply provide code which will actually run on the client. In fact, you can now run almost any Visual Basic program this way, using a UserDocument (VBD file), and similar facilities exist in other modern development environments, in which case many of the issues here are covered automatically for you, but for those wanting to make their own pages, this section covers a few important issues.

Needless to say, this method only works where you know that all the clients will be PCs running Windows, and preferably using Internet Explorer, so this is not a solution for everyone.

### 9.2.1 Creating the DicomViewer

Most modern web design programs will allow you to add ActiveX controls into your web page, and set their properties easily, but those who may be doing this by hand should use something like:

```
<object id="Viewer" name="Viewer"
classid="clsid:853AAF97-E49C-11D0-A303-0040C711066C"
align="baseline" border="0" width="800" height="768">
<param name="BackColor" value="3158064">
<param name="StretchToFit" value="-1">
</object>
```

Once such an object exists on your web page, you may reference it, just as you would any other DicomViewer, and the VBScript syntax can be used for events – e.g.

```
Sub Viewer_OnDataChanged()
...
End Sub
```

### 9.2.2 Issuing Queries

Because GetImages is a simple synchronous method, it can be used in a web client exactly as you would in any other application. The only concerns when issuing direct queries from a web page are security and validation, as most SCPs will answer queries only to a list of “known” SCUs with matching IP addresses. As a result, you may need to configure your SCU AET list to include all PCs from which web-based queries may be issued.

### 9.2.3 Retrieving Images

If C-GET is used, then the issues in retrieving images are exactly the same as for queries as above, but if C-MOVE is used (normally via GetUsingMove), then a few more issues are involved:

- The details of every SCU **must** be entered into the SCP, and this is still required, even if the SCP operates in promiscuous mode
- The web script must be able to determine its own AET for use as the C-MOVE destination. Possibilities for this include a variation on the IP address (returned perhaps from the web server to which it is available as a script variable), or the machine name [Environ(“ComputerName”)].

Of course, it is not strictly necessary to use DICOM network methods to retrieve the images – if the web server is set up appropriately, then direct http retrieval of DICOM files is possible, and may be more secure and efficient.

### 9.2.4 Security Issues

There is an ActiveX security system to prevent mis-use of trusted components by malicious scripts, and this does cause some problems for DicomObjects within a web page. In particular, the following actions need careful scrutiny:

- Reading and writing local files

- Network transfers

While the security implications of local file access are obvious, the network threat is more subtle, but imagine how a user with DicomObjects installed on their machine would feel if they visited a web page on the Internet which ran a script which:

- Searched for a local DICOM SCP
- Queried that for images (perhaps for a VIP)
- Retrieved those images
- Sent them to an SCP over the Internet
- Did all the above without the user even knowing anything was happening

The above might seem a little far-fetched, but most possible security holes have been exploited, and DicomObjects must not provide a gateway for malicious users. To prevent this type of thing, ActiveX controls can be “signed”, as DicomObjects is, with a full publisher’s certificate, and they must then be marked as either safe, or unsafe for scripting, but here is the problem:

- If DicomObjects were marked unsafe, then users could not run scripts using it unless they lowered their security settings, which is unacceptable in most healthcare environments.
- If marked as safe, then the above attack becomes possible.

i.e. it is impossible for DicomObjects to distinguish between intended and malicious use.

The compromise adopted in DicomObjects is to mark it as safe, but to notify the user the first time that any of the hazardous operations are attempted from within a web script. This is slightly intrusive, but considered worthwhile to preserve security. Customised versions of DicomObjects with variations on this policy are available on request, such as an unsigned version without the warnings, which might be suitable for use on a closed, entirely trusted network. Please ask for details if you think that you would prefer a different scheme.

### 9.2.5 Auto-Downloading

Just like any other ActiveX control, DicomObjects can automatically be downloaded into Internet Explorer on request, providing that a suitable CODEBASE parameter is included in the object definition. To help with this, Medical Connections can supply a full, signed CAB file for this purpose.

### 9.2.6 Licensing Issues

Once you use a registered version of DicomObjects, **all** DicomViewer controls need licensing information to be passed to them. In other environment, this is through the .lic file for design mode, or embedded in the application for execution mode, but in web pages, it is provided through a license pack (.lpk) file, which must contain the license information for all licensed controls on the page. Medical Connections can supply a ready-made .lpk file to developers if necessary, but it is generally better to make your own (which you can if you have the .lic file), in case you need to include any other components. See the following Microsoft knowledgebase article for more details:

<http://support.microsoft.com/support/kb/articles/Q159/9/23.ASP>

## 10 Writing a Router/Modifier

One of the common uses of DicomObjects is as an intermediary between other pieces of DICOM equipment, and in this rôle, the ability to receive data, make simple changes, and pass it on with relatively few high-level commands has proven particularly useful. Example include:

- Receiving data from primary equipment which fails to include to some attributes, adding these attributes, and then passing on to a server which regards them as mandatory (even if blank).
- Receiving images, then sending them on to one or more other destinations, depending on any of the attributes of the image, such as modality, requesting physician etc.
- Receiving data from primary equipment which provides faulty attributes (as some screening equipment does with patient names), using a modality worklist query to obtain corrected values, and making corrections before passing the images on to the server.
- Receiving images on several different associations, and passing on via a single association to servers which make the incorrect assumption of one association=one series.
- Channelling Q/R requests from multiple clients through a single machine to circumvent technical limits on number of connections.
- Providing transfer syntax conversion on either end of a slow link – perhaps connecting two pieces of equipment which do not, naturally support any of the compressed syntaxes.
- Passing Q/R request on to multiple other SCPs, and providing consolidated responses.
- Last, but far from least, due to DicomObjects' tolerance of the multiple known faults in other implementations, there are times when it can be used simply to receive data and pass it on essentially unchanged – a “DICOM to DICOM converter”!

The most basic of examples, adding a null element to an image and passing it on, is of course trivial in DicomObjects – assuming that a DicomViewer or DicomServer is listening on an appropriate port, then the following code will add a null requesting physician, and pass on another SCP:

```
Private Sub Viewer_ImageReceived(ByVal ReceivedImage As
    DicomObjects.DicomImage, ByVal Association As Long, isAdded As
    Boolean, status As Long)
    ReceivedImage.Attributes.Add &h8,&h90,""
    Status=ReceivedImage.Send(SvrNode, SvrPort, SvrAE, MyAE)
End Sub
```

Any more complex use will of course require code specific to the application, but a few general hints may be useful:

- For simple applications such as that above, it is easiest to use synchronous operation, completing one operation at a time.
- For more complex operations involving receipt from, or sending to, multiple applications, asynchronous operation using ImageReceivedAsync and DicomConnection objects may be more appropriate. In these cases, it is helpful to keep **all** operations asynchronous, and observe the notes in the server section (11.7.3 below) on avoiding inadvertent synchronous operation. In particular, it has been discovered that some SCPs cannot handle other operations whilst waiting for a C-FIND response, and this can easily lead to deadlocks if your own code has similar constraints.

## 11 Writing a DICOM Server

DicomObjects has now been used for several years as the lynchpin of several DICOM servers, and much of its functionality has been designed specifically around this rôle. The following features make it particularly suitable for this task:

- Robust design with no intrinsic message boxes or other GUI features to cause a server to require user intervention
- Extensive use of internal multi-threading features, enabling a fully-featured multi-tasking server to be written using a single-threaded high level language such as Visual Basic.

A very simplistic server is included with the examples in the download package, and this server runs entirely within Microsoft Access, using the intrinsic VBScript. Although this server lacks most of the features expected in a commercial system, it does demonstrate how simple writing such a server can be using DicomObjects. You are advised to look through the code of this server before attempting to write your own, if only to see how the components fit together. In general, however, this section assumes a greater knowledge of DICOM than elsewhere in this manual, on the basis that those contemplating writing a DICOM server are likely to have more than a basic knowledge of DICOM.

The sections in this chapter, first look at the requirements for a “normal” storage server, and then consider more general issues involved making your DicomObjects based server fast, efficient and reliable. Finally, other types of SCP are considered.

The main features required of DicomObjects-based servers are listed, but all authors are likely to have different specific additions.

### 11.1 An object to Listen for Associations

Whatever type of DICOM SCP is being written, it is necessary to listen on one or more TCP ports, and to react to associations received on that port. This requires an object to handle the listening and provide a “base” for the events that incoming activity generates.

There are two choices for this, a DicomViewer and a DicomServer. Functionally, they are almost identical for this purpose, and the choice depends more on personal preferences and support from your development environment than on intrinsic differences. The differences are:

- A DicomViewer is a control which can only exist within a window. In some environments this may be a problem, but generally it presents no problem even for NT services, as even services have a hidden window, and all windows and/or the controls on them can easily be hidden. Almost all development environments have good support for attachment of events to such controls.
- A DicomServer has all the same listening and response features, but does not have its own window, does not require a parent window, and cannot display images. It is therefore the logical choice for a server where possible, but many development environments provide very little support for attaching event routines to such non-visual objects, and where attaching events is difficult, a DicomViewer may be much easier to use.

Whichever object is used, it will be referred to from here on as “**server**” irrespective of its type.

The first requirement for any network server is to listen for incoming associations on a known port, and this is achieved in DicomObjects using the server’s Listen method, the most common port being 104. This is normally called as part of the application’s or form’s initialisation procedure. It is perfectly possible, if you wish, to listen on more than one port, in which case multiple Listen methods may be used. An equivalent Unlisten method exists, but is rarely used, as all active listening activity is automatically stopped when the server object is destroyed.

## 11.2 Validating Associations

When an incoming DICOM association request is received, the first event to fire is AssociationRequest. In this event, you should do the following:

- Validate the calling and called application entity titles, possibly together with the remote IP address, against a locally defined “allowed” list (this check is not normally performed during early testing). If the validation fails, then the association as a whole can be rejected by setting the isOK parameter to false. DICOM requires that every application should have an Application Entity Title (AET), which should be unique within a given network. DicomObjects does of course support this requirement, but in practice, associations are set up based on IP address and port number rather than AET. It is the programmer’s responsibility to check incoming AETs and to respond as desired. By default, unless over-ridden in AssociationRequest, DicomObjects will accept any association, irrespective of the AETs.
- Check the list of requested abstract syntaxes/SOP classes against the operations supported by the server. For instance, a server providing storage and Q/R services should reject printing abstract syntaxes. In general, any unofficial private syntaxes should also be rejected, as equipment from one of the major manufacturers will normally offer private syntaxes as well as official ones, and will use those private syntaxes in preference to the official versions unless the private syntax is rejected. To do this, reject any syntaxes not beginning 1.2.840.10008. If required, you may also choose the transfer syntax for each abstract syntax you accept at this stage, but if you decide not to do this explicitly, DicomObjects will make a sensible choice, based on either a preset list of priorities, or on a list stored in the registry. See section 17.3.4 for more details of this procedure. Pseudo-code for this whole process is as follows:

```

For each context in contexts
    If context.AbstractSyntax is acceptable then
        Choose TS=best value of those in context.OfferedTS
        Context.AcceptedTS=TS
    Else
        Context.Reject 3
    End if
Next context

```

## 11.3 Handling C-STORE Operations

When C-STORE operations are received, all reception activity occurs on the association’s own thread, without “bothering” the main program, but once reception of the images is complete, either ImageReceived, or ImageReceivedAsync will fire to announce its availability. ImageReceived will be reviewed first, as it is simpler, followed by a discussion of ImageReceivedAsync, which is really more appropriate for high performance servers.

### 11.3.1 ImageReceived

This is a very simple event, whether fired on a DicomViewer or a DicomServer, and your server should simply do whatever it wishes with the image, set the Status parameter to a suitable value depending on the acceptability of the image, and then return. Once the routine returns, DicomObjects will pass the status value back to the SCU, which may then either close the association, or carry out further operations. Of course, the main operations carried out by a server will be storing the image itself somewhere (see notes below on type of storage to use), and recording the demographics extracted from the image into a database for future Q/R operations. If using a DicomViewer as the server it is important either that its DefaultAdd property be set false, or that the isAdded parameter be set to false during this routine. This prevents the image being added to the viewer’s Images collection, which would rapidly deplete the available RAM. Although this event is simple to use, it does have a

significant disadvantage for server use, in that it is synchronous, so no other server operations can occur until the image has been saved. This may (e.g. for cardiological multi-frame images) be a long process, so for production servers, the following routine may provide better responsiveness.

### 11.3.2 ImageReceivedAsync

This routine is called in place of ImageReceived if the server's AsyncReceive property is true.

Unlike ImageReceived, the parameter list omits the Status parameter, but instead includes a DicomConnection object, which is the main object type used for asynchronous operations in DicomObjects. This enables the status to be sent at any time, using SendStatus, removing the requirement for the server to delay other operations until after it has saved this image. Threading issues are discussed in more depth below, but it is worth noting at this stage how useful the SaveImage method (of DicomConnection) is for this purpose, as simplified pseudo-code for efficient storage is as follows:

```

Server.ImageReceivedAsync(Connection, Image,...)
    [determine where to store image, and update database]
    Connection.SaveImage(Image, Destination, ...)
Return

Server.ActionComplete(Connection, Action, Tag, Success, ErrorMessage)
    If Action="SaveImage" then
        If Success then
            Connection.SendStatus(0) // Success
        Else
            Connection.SendStatus(0xC000) //Or more specific fail code
        End If
    Else
        ...
    End If
Return

```

The result of this code is that actual writing of the image occurs in the background on a separate thread (one is associated with every asynchronous DicomConnection). During this time, the SCU that sent the image is kept waiting, but the server can service other operations while the save is in progress. Only when the save is complete is the attention of the main thread required, to send the status code to the SCU.

## 11.4 Handling Query/Retrieve Requests

This is the most complex part of writing a good DICOM server, but using DicomObjects you should be able to concentrate on the functionality, and leave most of the mechanics to DicomObjects itself. In general, you have 3 or 4 variables to consider:

- **The type of query**  
(“Operation” property of Connection)
- **The “Root” of the query**  
(Connection.Root)
- **For C-FIND requests only, the “level” of the query**  
(attribute 0x0008,0x0052 of Connection.Request)
- **The dataset to match**  
(Connection.Request)

Whatever the type of the query, you will need to need to use the root and the dataset to query your own database, and produce a list of matching items. A simplistic example is provided in the Access

example server, but in a production server, the SQL or similar queries are likely to be more complex. Note that for C-FIND requests, the data to return may represent patients, studies, series or images depending on the “level” of the query, but for C-GET and C-MOVE, the requirement will always be to locate images themselves.

In the sections below, a sequence of calls is specified. Whilst these may, especially for testing, be made consecutively, please note the comments in section 11.7.3 below concerning the placement of these calls for best operation.

### 11.4.1 C-Find

This section deals with image-orientated C-FIND operations – modality worklist queries also cause this event to fire with operation=C-FIND, but the root value will be WORKLIST, and such queries are dealt with in section 11.5 below.

Your task is conceptually quite straightforward, but details will vary considerably according to the database used to store image details. You should:

- Construct a query using as many as possible of the non-null values in the query dataset.
- Arrange for the returned data to include valid attributes corresponding to as many as possible of the items in the query dataset.
- Construct a new DicomDataSets collection object.
- For each match located in the database query, construct a new DicomDataSet object, fill it with the required values, and add that dataset to the collection created above
- Call SendData using the collection, and an appropriate status value. If all requested items in the query dataset have been used for matching, the status should be 0xFF00 , but if some are not supported, then the status should be 0xFF01.
- If required, multiple SendData calls may be made, but a single call is generally more efficient.
- Finally, a call to SendStatus with a value of 0 will return that value (success), and complete the operation.
- If an error occurs, and the query cannot be processed, a suitable status code (in practice anything other than 0xFF0? or 0) will terminate the operation. Where possible “official” status codes should be used.

The following features of DICOM queries may require special consideration and handling:

- All attributes used for matching should be returned in the result datasets.
- If an attribute is sent with a null value, then it should not be used for matching, but the corresponding value should be included in the results.
- DICOM uses the \* and ? characters to indicate wildcard matching.
- Items which are normally defined in DICOM to have single values, such as PatientID and instance UIDs may, in requests only, have multiple values, which are to be interpreted as “or” matches. However, as such attributes have a normal VM of “1”, DicomObjects will supply them (via the Value property) as single strings, the components being separated by \ characters. e.g. a query requesting details for patients with IDs 12345 and ABCDE will give the string “12345\ABCDE” when either the PatientID property or Attributes(0x0010,0x0010).Value is requested. It is the server programmer’s responsibility to interpret such strings, and construct suitable database queries.

- DicomObjects will normally provide date and time values as variants of type VT\_DATE, but DICOM allows date and time ranges in queries, such as:
  - 20010101-20010103            1<sup>st</sup> to 3<sup>rd</sup> January 2001 inclusive
  - 20010101-                    Any date from 1<sup>st</sup> January 2001 onwards
  - -20010103                    Any date up to and including 3<sup>rd</sup> January 2001
  - similar formats for times

Such ranges cannot be represented as variant dates, so are normally returned by DicomObjects as strings exactly “as received”. It used to be the programmer’s responsibility to interpret such ranges, and construct suitable queries, but as of version 4.1, two new properties have been introduced DateTimeFrom and DateTimeTo, which greatly simplify the handling of such queries – please consult the help file for more details.

### 11.4.2 C-GET

There is a general move in the industry away from support from C-GET, and most recent applications only support C-MOVE. None the less, C-GET is easy to support using DicomObjects, and is simpler than C-MOVE, with many similarities, so this is demonstrated first, with the necessary changes for C-MOVE following.

In fact, the necessary programming for C-GET is almost identical to that for C-FIND as above, except that the data required is always images, so when the “level” is anything other than IMAGE, a relational search will always be required. However, having located the images you should send, the implementation is actually easier than for C-FIND, as you only have to send the images themselves, without having to worry about which data attributes were or were not requested in the query dataset. The method to send the images is SendImages, but this method in itself has many possible ways of specifying the image source, these being summarised below.

- **A DicomImage object**  
The single image is sent.
- **A DicomImages collection**  
All images are sent.
- **A String**  
The file referenced by the string is read and sent.
- **An array of strings**  
All are interpreted as filenames, which are read and sent.
- **An object supporting IStream**  
The stream is read (as for ReadStream) and is sent.
- **An ADO, DAO or RDO field object**  
The field is read (as for ReadField) and is sent.

Of these possibilities, the first two are the most intuitive, and they are also the most flexible, as the image may modified “en route” if necessary (e.g. to coerce demographics etc.). This approach is also useful if the images have been fetched from another DICOM server. For a high-performance server however, the string parameters provide much better performance, as the reading of the file from disk occurs on the DicomConnection’s own thread, *not* on the main thread, as would be the case if ReadFile were used, followed by SendImages with a DicomImage(s) parameter. The final two sources are rarely used in practice, though stream representations of images can be useful in some environments (especially Delphi).

If you wish to check the result codes returned in response to sending images, you can check the connection’s Status property, which will return an array containing one value for each image sent, but

please note that this property is only valid once the operation has completed (i.e. when the `isReady` property is true, after the `Wait` method, or in an `ActionComplete` event).

As with `SendData`, this method may be repeated as required, and is followed, once all images have been sent by `SendStatus(0)`.

Note that you can only send images for which the SCU has proposed, and your server has accepted, a suitable presentation context. So, you must always be prepared, when sending images, to receive error 15 (Attempt to use Abstract syntax (xxx.yyy.zzz) not in negotiated list).

### 11.4.3 C-MOVE

Although the underlying mechanisms used in DICOM are very different for C-GET and C-MOVE, `DicomObjects` has been written to simplify support for both, by providing a common interface, namely `SendImages` as above. However, one important addition to the requirements for C-MOVE is the need to translate the destination AET provided in the request (`Connection.Destination`) to a combination of IP address and port number, as DICOM itself demands that these be kept in a server database, and there is *no* way for these to be transmitted within the request itself. Once the server has done this lookup (and presumably found a valid result), it should call `SetDestination` using the values obtained. Once this has been done, images are sent, and the operation terminated exactly as described above for C-GET. If the destination is unknown, the query should call `SendStatus(0xA801)` and exit.

One difference that you may need to consider is that there needs to be a valid presentation context on which to send the images, which requires that your server should propose all necessary syntaxes (and hope that the recipient accepts them). By default, `DicomObjects` uses a list which includes all commonly used image SOP classes, but if you are sending unusual SOP classes (such as structured reporting objects, or radiotherapy objects), you will need to ensure that suitable contexts are set up. There are two ways of doing this:

- Add them to the `AbstractSyntax` registry key
- Add them explicitly by using `DicomConnection.Contexts.Add` before calling `SetDestination`.

In either case, you will be over-riding the default list entirely, so you need to ensure that all other SOP classes required to be sent are included. If the logic of the server permits, the ideal would be to check the SOP classes of the images to be sent and ensure that all are added explicitly before calling `SetDestination`.

### 11.4.4 Progress Notifications

DICOM provides a mechanism for sending progress notifications during C-GET and C-MOVE operations, and though these are optional, your server may support them if you wish by inserting `SendStatus(0xFF00)` calls between `SendImages` calls. Five properties of the `DicomConnection` are passed to the SCU as part of such a status call, these being:

- `NumberRemaining`
- `NumberCompleted`
- `NumberWarning`
- `NumberFailed`
- `Failures`

By default, `DicomObjects` will, at the start of `SendImages` set `NumberRemaining` to the number of images to be sent by that call. The other numbers are initially zero, and are incremented appropriately as the operation progresses, depending on status codes received. Likewise, `Failures`, a list of failed SOP instance UIDs will be updated automatically as necessary. Any of these can if needed be over-ridden, but the only common requirement is to set `NumberRemaining` to an appropriate value when

more than one SendImages call is to be made. In this case, DicomObjects will leave your initial value untouched if it is greater than the number of images to be sent by that call, but the auto-decrementing will still, of course, occur.

### 11.4.5 Error Handling

DICOM's main error notification mechanism is the status code, and a server can terminate a failed operation by returning any status other than pending (0xFFxx) or success (0) using SendStatus. After such a call, no further action is needed or possible, and provided that the status code is appropriate, this should be handled correctly by the SCU. It is often useful however to provide more information about the error, which may be displayed or logged by the SCU, and for this the connection's Errors property may be used. This is a DicomDataSet, and the following values may usefully be added to it when an error occurs:

- (0x0000,0x0901) : Offending Element
- (0x0000,0x0902) : Error Comment

Only the appropriate elements, as defined by the returned status will be transmitted.

These values should be set up *before* calling SendStatus with an appropriate error code.

## 11.5 Handling C-ECHO Requests

The routine to handle C-ECHO request is the simplest possible – it should set the Status to zero, and apart from any internal logging you may require, it should do nothing else. The importance of this routine is that the status would remain at the DefaultStatus value (normally non-zero) if, for any reason, event routines are not being called.

## 11.6 Transfer Syntax and Quality Issues

For C-GET, you will not normally need to worry when sending images from a server about abstract and transfer syntax issues, as it is the responsibility of the SCU to propose suitable presentation contexts. Similarly, when using C-MOVE, the default list of SOP classes/abstract syntaxes cover most images, but if you are using something new or unusual, you may need to ensure that suitable contexts are proposed (see section 4.8 above for example code to do this). Assuming that one or more suitable presentation contexts have been negotiated, DicomObjects will pick a suitable one for the transfer, based on the SOP class of the image being sent without further assistance, but there are occasions where more precise control may be required, and the following sections deal with three such situations:

### 11.6.1 More than one Suitable Presentation Context

By default, where more than one presentation context is available for a given SOP class, DicomObjects will choose the first (lowest numbered) in the negotiated list, but if you wish, the PreferredTS and PreferredPCID properties of the connection may be used to express a preference, subject always to a suitable presentation context being available. See the help file entry for the above properties for fuller details of the priority system used.

### 11.6.2 Lossy Compression Quality

To control the quality factor used when compressing images “on the fly” call the SetQuality method of the connection, passing both the quality, and the transfer syntax it is to apply to. For lossy JPEG images, the quality factor ranges from 0 to 100, but other private schemes may have different scales. Note that DicomObjects does not unnecessarily decompress/recompress images, so if the images were received or read in a suitable matching compressed form, then this value will be ignored. To force recompression, read the images into RAM, and use the DecompressAll method before sending (though multiple lossy compression is not generally considered good practice).

### 11.6.3 CoercionSOP

Normally, if a suitable presentation context has not been negotiated for a particular image, then attempts to send it using SendImages will fail with a suitable error, but as a lowest level fallback, DicomObjects will, if requested and if no proper presentation context exists, change the SOP class of an image to the value held by the CoercionSOP value of the connection and try again. The default value is null (no coercion allowed), and the only sensible other value is 1.2.840.10008.5.1.4.1.1.7 (secondary capture).

## 11.7 Performance and Reliability Issues

### 11.7.1 DefaultStatus

It is vital that the DefaultStatus property of a server object used in a production server should be non-zero, and should be a value indicating a generic error (0xC000 is commonly used). All events handling operations successfully should of course over-ride this to 0, but if for any reason these events are not called (e.g. as can happen in Visual Basic if a message box is active), then the SCU is at least informed of the error rather than being sent an incorrect “success” value.

### 11.7.2 How to Store your Images

DicomObjects provides the facility for images to be stored in multiple forms, including database fields (DAO, ADO or RDO), Streams, or files. Whilst the database formats are useful for small examples (and are used in the Access server example), most commercial databases (especially Microsoft SQL) handle “BLOB” data very badly and inefficiently, so you are strongly advised to use conventional files, storing only the filenames etc. in your database.

### 11.7.3 Threading Considerations

The server capabilities of DicomObjects are written in such a way that it is possible to write a full multi-threaded server from a single threaded environment. In order to achieve this however, it is necessary to follow some rules for “production” use. The important point to remember is that a DicomConnection, even when in asynchronous mode (as every incoming request always is) can only have a single outstanding operation, so any further operations will cause the calling thread to wait until the previous operations have completed. This is best shown by an example, assumed to lie within a QueryRequest event, where Connection is the DicomConnection provided as a parameter to the event.

```
Connection.SendImages("C:\images\image1.dcm")
Connection.SendStatus(0)
```

When this code is executed, the SendImages method will return immediately, and transfer of the images on the established association will commence. However, the SendStatus method cannot return until the operations initiated by the previous method have completed, and this may be a long time, during which the server would be unresponsive to other requests (and could, if similar problems exist in the SCU, lead to a deadlock) so this needs to be avoided. To avoid this issue, each operation should be initiated only once the previous one has completed, and the best way of doing this is to use the ActionComplete event. Using this approach, only the very first operation is triggered from with the QueryRequest event, and all other operations are called from the ActionComplete event. Pseudo-code for this would be:

```
server.QueryRequest(connection,...)
  if operation=C-MOVE then
    [lookup destination]
    connection.SetDestination(...)
  else if operation=C-GET
    [lookup images to send]
```

```

        connection.SendImages(...)
    else if operation=C-FIND
        [lookup image details]
        connection.SendData(...)
    end if
return

server.ActionComplete(connection,...)
    if success then
        case(Action)
            SetDestination:
                [lookup images to send]
                connection.SendImages(...)
                break
            SendImages:
                connection.SendStatus(0)
                break
            SendData:
                connection.SendStatus(0)
                break
            SendSatus:
                connection.Close
        end case
    end if
return

```

Although not used in the above simplistic example, there will often be occasions where the routine initiating an asynchronous method needs to pass some data or context information on to the ActionComplete event, to ensure that subsequent operations are appropriate. The Tag property is designed for this purpose. Note that the value passed to the ActionComplete event is the value at the time that the initiating method is called, so Tag should be set *before* making any such calls.

For testing and demonstration the requirements are less strict, and it may be easier just to concatenate the calls, (as is done in the Access viewer for clarity), but this once the server is working, it should be “dissected” as above for production use.

### 11.8 Modality WorkList SCP

From a simple DICOM point of view, modality worklist (MWL) queries are structurally the same as other C-FIND queries, i.e. a query dataset is received, and matching datasets are returned through SendData. There are however a few important differences:

- The “Root” property is set to WORKLIST.
- The results represent scheduled examinations rather than patients, studies etc.
- Most of the details of the request, and most of the results, are placed within the scheduled procedure steps sequence (0x0040,0x0100) within the main dataset.

Apart from the above differences, the basic methodology is just the same as for other queries – extract query elements from the dataset (mainly from the sequence) and run a query against the Radiology Information System database. The procedure step details should of course be embedded in a single-item dataset.

### 11.9 Print SCP

It is possible, using DicomObjects to write a print server, an SCP which receives the normalised operations used in DICOM printing, responds appropriately, and does something useful with the

images ultimately received, such as print to a Windows printer. Such operations would be mediated through code in the NormalisedReceived event, and all data sent is readily available for this purpose. Before embarking on this type of project however, two important points should be borne in mind:

- The print protocols in DICOM are extremely complex, and your “pseudo-printer” would need to create internal representations of objects such as films and image boxes as requested by the SCU.
- The image you would eventually receive would consist only of pixel data and immediately associated attributes such as image dimensions and pixel depth. No associated demographic data is normally present, other than that “burnt-in” the pixel data, so the images are not generally suitable for other purposes.

### **11.10 Storage Commitment SCP**

DicomObjects can be used for storage commitment, but this is an advanced technique requiring new procedures such as reverse SCP/SCU rôle reversal, and it is considered in a separate section (17.6 below).

## 12 Accessing and Modifying Pixel Data

There are several ways to access and modify the pixel data within a DicomImage, but the best method will depend on your environment, in particular whether the language, such as C++ supports raw data pointers or whether like Visual Basic, it has good support for variant arrays (SAFEARRAYs).

### 12.1 Languages with Raw Pointers

When using a language such as C++, by far the easiest way to access pixel data is to use the PixelAddress property, which should (subject to the note below on compressed data) give you a pointer to the start of the actual data. This will always be in little-endian form (DicomObjects does any necessary conversions while reading/writing), and should be interpreted as a pointer to either bytes or short integers according to the values of bits allocated (attribute 0x0028,0x0100). This data is completely unprocessed, so if you do any calculations on it, you may need to allow for DICOM variables such as:

- Samples per pixel (3 for colour images)
- Photometric interpretation
- Pixel Representation (whether data is signed)
- Planar configuration (RGBRGBRGB or RRRRR... GGGG... BBBB...)
- Rescale parameters

In a multi-frame images, frames follow on consecutively, so you will need to calculate the frame size as  $X \times Y \times \text{Bytes}$ , and index accordingly.

The value returned by PixelAddress is actually a long integer, which just needs casting to an appropriate pointer – this is because some languages incorrectly try to dereference official OLE pointers, and just pass a single byte or short!

If the data is writeable (see below for a possible pitfall), then it may be freely modified through this pointer. Once the modification is finished however, it is necessary to use the PixelsModified method to inform DicomObjects that the image needs to be re-cached and redisplayed.

In general, DicomObjects tries to conserve RAM, and so it reads and decompresses data only when necessary. This has two effects for this type of data access:

#### 12.1.1 Compressed Data

Normally, DicomObjects only decompresses compressed data as and when necessary, so if a compressed image has been received or read then no large single block of decompressed data will exist. If therefore you attempt to access PixelAddress for an image currently held in compressed form, an error will be generated. To avoid this, you can use the image's DecompressAll method, which will force all frames to be decompressed at once. Beware, on large angiography multi-frame images, this can require a large amount of RAM!

#### 12.1.2 Data Read from Disk

The pointer obtained using PixelAddress is normally to read/write RAM, enabling modifications as above, but when images are read from disk using ReadFile, the default action is to map the pixel memory to the file itself, as this saves RAM, especially for large multi-frame runs. However, to prevent inadvertent file modification, this is done as read-only access. To circumvent this, if write access is required, call the image's DecompressAll method before obtaining the pointer, just as for compressed data. This will force the data to be read into RAM.

## **12.2 Languages Using Variant Arrays**

For languages which do not support raw pointers, DicomObjects can supply a copy of the entire pixel data in a SAFEARRAY. This will be of the appropriate data type (byte or short integer), and will have three dimensions, x, y and frame, even where there is only a single frame. By its very nature, this type of access is likely to be much slower than using a raw pointer, and for multi-frame images, a large amount of RAM may be required.

If you wish to modify the pixels, a compatible array can be used, and simply assigned to the image's Pixels property. For this purpose, a 2 dimensional array is acceptable for a single-frame image.

The constraints listed above for raw pointers, concerning compressed and disk-resident data do not apply to this method of access.

Where samples per pixel is >1, the format will vary according to the planar configuration:

- 0: The x dimension will be increased (normally by a factor of 3), and the values will appear as R1 G1 B1 R2 G2 B2 etc. along each line.
- 1: The y dimension will be increased (normally by a factor of 3), and all the lines for each component will follow all the lines for the previous component.

## 13 Creating DICOM Images

DicomObjects has extensive facilities for the import of images from other formats, and the formats with native support in DicomObjects are BMP, DIB and JPEG. Creating an image from these formats is easier than making one “from scratch”, so this will be examined first, followed by multi-frame imports, and finally, complete image creation without an existing file.

As of Version 4.1, AVI files may also be imported using the FileImport method, which will generate a multi-frame DICOM image.

### 13.1 Importing Other Formats

Before going further, some important points need to be made, as these constitute some of the most common mistakes made using DicomObjects.

- The Import methods, like any others, need an existing object (in this case a DicomImage) to work on.
- DicomObjects cannot guess what type of image (CT, MR, visible light etc.) you are creating, so you must explicitly set the SOP class.
- Other data will be required to make a complete and valid DICOM object, understandable to other software.

Using them these points, there are 6 stages to creating a valid DICOM image:

#### 13.1.1 Create Your Image

Use your language’s “new”, CreateObject or similar call. e.g. in Visual Basic this would be:

```
Set image=new DicomImage
```

#### 13.1.2 Import the Pixel Data

For full details, see the FileImport, MemoryImport or ArrayImport commands in the help file. A typical use would be:

```
image.FileImport "C:\testimage.jpg", "JPG"
```

#### 13.1.3 Choose the SOP class (Abstract Syntax)

This value, which is a standardised DICOM UID is normally chosen from the list available in the DICOM standard. In many cases, imported images will be the “lowest common denominator” in DICOM, the secondary capture format, but this will not always be the case. The attribute to hold this value is 0x0008,0x0018, so the VB code to define a secondary capture object would be:

```
image.Attributes.Add 8,&h16,"1.2.840.10008.5.1.4.1.1.7"
```

#### 13.1.4 Add Other Data Necessary

The other data necessary will vary from one SOP class to another, and the requirements are listed in the information object definitions in the DICOM standard. Some of these are mandatory (e.g. the patient’s ID), some are optional, and others (including in fact the patient’s name) are “type 2”, which means that they must be present, even if blank (if genuinely unknown). These are all added individually, using a format similar to the following VB pseudo-code:

```
image.Attributes.Add group,element,value
```

e.g. to set the referring physician’s name to “Dr Smith”, the code would be

```
image.Attributes.Add 8,&h90,"Dr Smith"
```

You will probably need between 10 and 30 such lines depending on the SOP class you have chosen, and the amount of data available to you. In general, as much data as is available should be added to the image.

Some attributes, such as the patient's name, may of course be set via shortcut properties, but the effect is exactly the same as using the above syntax.

### 13.1.5 Set Your UIDs

All DICOM images must have unique instance UIDs. In addition, each series and study must have unique UIDs (which must not be the same as instance UIDs or each other either). When a new image is created, DicomObjects helpfully creates all of these for you, virtually guarantees them to be unique, and this guarantee is absolute if you use a unique root for the machine (see UIDs in the help file for more details). However, DicomObjects does not know how the images should be arranged into series and/or studies, so if you need this type of arrangement, you must modify the images' SeriesUID and StudyUID properties to match. The easiest way to do this is to store the series/study UID of the first image of a series/study, and then use that to replace the relevant UIDs in subsequent related images.

### 13.1.6 Write Your Image to Disk

Even if you intend finally to send images over a network, it is probably easiest for testing to write to disk, using the WriteFile method. Note that DicomObjects does automatic conversions between transfer syntaxes (formats), so any type of image, however imported, may be written in any format. Special handling however occurs where an image which is already lossily compressed is written out or sent in a compatible format, as the image is not decompressed and recompressed. This is triggered where an imported JPEG image is written out using a JPEG transfer syntax (1.2.840.10008.1.2.4.50 or 1.2.840.10008.1.2.4.51), and it has two notable features:

- There is no further loss of image quality.
- The "quality" parameter to the WriteFile command (and network equivalent) is ignored.

### 13.1.7 Test Your Image

While DicomObjects allows you to create absolutely any form of DICOM data object, it has no knowledge of the information object definitions required for a valid image, as this is a huge task which would require a separate program. Fortunately such a program exists, and it is free – the DVT diagnostic test tool. At the time of writing, this can be obtained from <http://www.dvtk.org/index.php>. The program is excellent, and thorough, but is difficult to use initially, so here is brief guide for this purpose:

- Install and run the tool.
- Select File | Open.
- Browse to and open C:\Program Files\DVT\Example\example.pdvt
- Double-click on Media.ses (left hand column)
- Browse to and open your DICOM file.

This will give you a comprehensive listing of your file, with any errors highlighted, for you to correct.

## 13.2 Importing Multiframe images

Most of the work involved in importing a multi-frame image from a series of individual images is similar to that above, but a few extra stages are required:

- Create a new DicomImages object.
- Import each image into a new DicomImage object, and add each image to this collection.

- Use the MakeMultiFrame method to create a single multi-frame image.
- Add additional attributes as described for single frame images.

The MakeMultiFrame method completes most of the attributes necessary for a multi-frame image, but it knows nothing of the relationship between the frames, which could for a nuclear medicine image be any combination of time, angle, photon energy etc., so it is still necessary to add the frame increment pointer (0x0028,0x0009), and any associated data in order to complete a valid SOP instance.

### 13.3 From Scratch

Before making images from scratch, it is a good idea to make some using imported image data, as all the stages described above are required **plus** new items to provide the pixel data.

When creating image from scratch, it is necessary to include the following attributes to describe the pixel data:

Tag (hex)	Description	Common Value(s)
(0x0028,0x0002)	Samples per pixel	1 (mono) 3 (colour)
(0x0028,0x0004)	Photochromic Interpretation	MONOCHROME2 (mono) "RGB" (colour)
(0x0028,0x0008)	Number of frames	[only required if >1]
(0x0028,0x0010)	Rows	
(0x0028,0x0011)	Columns	
(0x0028,0x0100)	Bits Allocated	8 or 16
(0x0028, 0x0101)	Bits Stored	8 or 12
(0x0028, 0x0102)	High Bit	7 or 11
(0x0028, 0x0103)	Pixel Representation	0 (unsigned data) 1 (signed data)

Several Methods exist to supply the pixel data itself:

#### 13.3.1 The Pixels Property

The easiest (though not necessarily the most efficient) method is to hold the data in a 2 or 3 dimensional array of bytes or short integers. In Visual Basic, this is a standard "array", but in other languages, it will normally be referred to as a SAFEARRAY or similar. To use this method, simply place your pixel data in the array, and then assign the array to the image's Pixels property. You need to ensure, of course, that the data you are providing is compatible with the descriptive values you have supplied as above.

#### 13.3.2 PixelAddress

Using this property is normally more efficient, and certainly easier to use from languages such as C++ which support simple pointers. To use it, allocate the correct space for your data using AllocatePixelSpace (remembering to allow for 2 or 3 bytes per pixel where necessary), then access the PixelAddress property, and cast to a pointer to bytes or short integers as appropriate to your data. The data may then be copied in or created using whatever method you prefer.

### **13.3.3 CopyPixelBuffer**

This method like the one above requires use of `AllocatePixelSpace`, but after this has been used, the `CopyPixelBuffer` may be used to move data efficiently from one location to another. You have considerable flexibility to define the row and column spacing of the original data, but the DICOM spacings are taken from the attributes defined above (which must therefore be defined before this operation occurs). This method is particularly useful for copying from video buffers for frame-grabbing applications written in interpreted languages, and makes a special case for colour data, swapping the component order from BGR (the Windows norm) to RGB as required by DICOM.

### **13.3.4 Mapping Data to a File**

If data already exists in a file, then this method may be used to “map” the data into the `DicomImage` without reading the whole file into RAM. This method is particularly useful when long multi-frame images are being created, as it massively reduces RAM usage by reading each frame only as it is used, and then discarding it. For this purpose it is best to set the `DicomImage`’s `CacheDisplay` property to `false`.

## 14 Using Modality WorkList as an SCU

This section is deliberately placed immediately after image generation, as the two are often combined in primary imaging equipment. By using this facility, equipment can retrieve details of scheduled examinations, reducing the need for operators to enter demographics and other examination data. This also helps to reduce errors due to mis-typing, and is expected in most new equipment.

Queries are normally made using a DicomQuery object in a similar method to “normal” queries, but due to the multiple variables involved, DoRawQuery has to be used, in place of DoQuery. One of the main oddities of using MWL is that some of the relevant information is, for both the query and the responses, contained within the Scheduled Procedure Step Sequence (0x0040,0x0100).

This is still however relatively straightforward, as shown in the example below, which retrieves details of all scheduled CT examinations today:

```

set query=new DicomQuery
[set Destination, Port, CalledAET and CallingAET]
set dataset=new DicomDataSet
set sequence=new DicomDataSets
set step=sequence.AddNew
dataset.Add &h40,&h100,sequence

dataset.Attributes.Add &h0040,&h0002,Now() ` specify today's date
dataset.Attributes.Add &h0008,&h0060,"CT" ` specify modality
[etc.]

step.Attributes.Add &h0010,&h0010,"" ` request name
step.Attributes.Add &h0010,&h0020,"" ` request ID
step.Attributes.Add &h0008,&h0050,"" ` request Accession
[etc.]

results=query.DoRawQuery(dataset)

```

Information will then be found in the individual datasets within results, some of it within the (0x0040,0x0010) sequence.

There are many variables which can be used and/or requested, and you are advised to check section K.6.1.2 of Part 4 of DICOM (2000 edition) for more details.

## 15 Language Specific Features

### 15.1 Visual Basic

Although ActiveX objects are theoretically language neutral, there is no doubt that they and Visual Basic were developed to work together, so almost every feature of DicomObjects works easily with Visual Basic, and few problems occur. Full details of all objects can be found in the browse window by pressing F2.

Note that some installations of VB have problems with registration of some of the common ActiveX components, and these can cause errors when the VB example programs are run. For a fix for these issues, please see the following Microsoft article:

<http://support.microsoft.com/support/kb/articles/Q177/7/99.ASP>

### 15.2 VBScript

In most respects, VBScript and Visual Basic are similar, so ActiveX objects such as DicomObjects are easy to use in this environment, but there are a few important differences:

- VBScript does not have the equivalent of the project references list, so objects cannot be referred to via shortened names, and all objects are simply dimensioned as “Object” rather than having a more specific type. Also, there is no “new” method – and CreateObject (or Server.CreateObject in ASP), should be used instead, remembering to use the fully qualified class name, e.g. DicomObjects.DicomImage.
- The only general variable type in VBScript is variant, and this has a subtle effect. The values returned from DicomAttribute.Value are always themselves variants, but they may contain arrays of other types, and this will produce errors if you try to access individual items within the array. To avoid this, replace the explicit or implicit Value properties by VariantValue, which will return values guaranteed to be variants, or arrays of variants. The same applies to the Pixels property, which can be replaced in this environment by image.Attributes(&h7fe0,&h0010).VariantValue.
- Unfortunately, there is no way to “embed” the licensing information held in DicomObjects.lic into VBScript, which therefore must always run effectively in “design” mode. As a result, the conventional methods of licensing control do not work, as any user with access to the machine has access to the license file, which could be mis-used. Therefore, for this type of application, a special version of DicomObjects has been made, which relies instead on other mechanisms – please ask Medical Connections for details.

### 15.3 Visual Basic for Applications (e.g. MS Access)

In general, the facilities are the same as for full Visual Basic, but like VBScript above, there is no facility for storage of the licensing information within the application, so if you intend to distribute applications using DicomObjects in this environment, you should request the version with alternative control mechanisms.

### 15.4 Microsoft Visual C++

Microsoft have provide two completely different mechanisms for use of COM objects in C++, MFC wrappers, and the #import directive, and each has its own characteristics. In addition, the attachment of events to the DicomServer needs to be considered.

Whatever form of wrapper is used, it is important to remember that an explicit call to CoInitialize or CoInitializeEx needs to be made once per program (in application.InitInstance or similar).

### 15.4.1 Import via MFC Wrapper

This mechanism is invoked (in Visual Studio 6) using Projects | Add To Project | Components and Controls, and the result is to add a series of header and code files, defining a new class for each of the objects you choose to import from the chosen library (presumed to be DicomObjects). These are named CDicomViewer, CDicomImage etc., and the main features are:

- This is an excellent method for the DicomViewer, as it can easily be added to dialogs, and attachment of events is easy using ClassWizard.
- A disadvantage is that it is a one-time static import and subsequent upgrades or changes to DicomObjects interfaces will not be visible unless the procedure is repeated.
- The objects created (CDicomXXXX) can create reference counting problems, especially if a dispatch pointer is assigned to an object which is then deleted, as this causes a Release without an AddRef. To avoid this, you may need to set the object's m\_bAutoDelete property to false.

### 15.4.2 #Import

This directive was introduced as part of the Active Template Library (ATL) features of MSVC++, but is useful even in MFC programs. It creates objects called IDicomImagePtr etc., which are reference counted encapsulations of the underlying dispatch pointers. Note in this context that IDicomImage\* is totally different from IDicomImagePtr. The main features are:

- The files produced (.tlh & .tli) are updated automatically if the DicomObjects OCX is changed.
- The wrappers have good reference counting, and easy access to all properties and methods.
- Although a IDicomViewerPtr object is defined, its use is not as easy as a CDicomViewer, and in particular, even if a window is created, attaching events is non-trivial (see next section of DicomServer events).
- To create “new” instances of other objects use the wrappers' CreateInstance method – e.g.:
 

```
IDicomImagePtr image;
Image.CreateInstance("DicomObjects.DicomImage");
```
- There is a bug in MSVC++ which will cause a compilation error when version 4.1 or above of DicomObjects is #imported. This can be avoided by putting the following line before the #import directive

```
#define FontPtr IFontPtr
```

### 15.4.3 DicomServer Event Sinks

As in most languages, the weakest part of COM support in MSVC is attaching events to non-visual objects such as DicomServers. This can be done by explicit use of event sinks, and this is demonstrated in the supplied MSVC example.

### 15.4.4 Trapping Errors

The wrappers created using #import check for errors reported by throwing `_com_error` exceptions if any errors exist. These exceptions may easily be caught using try...catch (not the structured exception handlers TRY..CATCH..END\_CATCH), and they contain both the error code (as WCode()), and the description(Description()).

### 15.4.5 Recommendations

My personal choice when using COM in MSVC++ is to use the MFC wrapper for visual objects (e.g. DicomViewer), #import, making IxxxPtr objects for other objects, and to avoid sinks on non-visual objects where possible (a DicomViewer can often be used in place of a DicomServer). All these methods are shown in the MSVC++ example, which you are advised to study.

## 15.5 Borland Delphi & Borland C++ Builder

Borland provides similar support for ActiveX/COM components in Delphi and C++ Builder, which will therefore be discussed together. ActiveX is supported through import routines, which create “packages”, which then give access to the objects, properties, methods and events. In previous versions of DicomObjects, a “package” was supplied, but this was found to cause more problems than it solved, and as of version 4.1, users should “import” the objects themselves. To import DicomObjects, select from the menu Component, then Install ActiveX Control, then choose DicomObjects, and finally hit the “Install” button (not build unit).

### Important Note:

There are serious bugs in the import routines in the original release of version 6 of Delphi, which will cause compilation to fail if you import yourself using these versions. These bugs are fixed in the latest updates, which are highly recommended.

Borland applications appears to have problems upgrading ActiveX controls properly. If after a DicomObjects upgrade you are experiencing unusual behaviour from a DicomViewer control, please try deleting the control and replacing it with a “new” one. If you do this, you must then remember to re-link the events on the control’s property sheet.

Note that the following words used in DicomObjects are reserved in Delphi, and have been replaced in this standard import file as follows:

Label	replaced by	Label_
Type	replaced by	Type_
Array	replaced by	Array_
XOR	replaced by	XOR_

The ItemByIndex property has been included to allow the members of a DicomAttributes collection to be iterated using a simple integer counter.

There is a bug in the Borland import routine in versions prior to version 6, such that properties and methods of a visible control (here a DicomViewer) which themselves return ActiveX objects (i.e. IDispatch pointers) are not properly imported. The most common property affected is the “Images” collection of a DicomViewer. To circumvent this omission, you should use the DicomViewer’s “ControlInterface” property, and access other properties from that. e.g. instead of:

```
DicomViewer1.Images
```

Use instead:

```
DicomViewer1.ControlInterface.Images
```

The easiest way to create new objects is to use the CoDicomXXX.Create method, as below:

```
Query:=CoDicomQuery.Create;
```

Attaching events to a DicomServer is not easy, but can be done. If you need assistance, please contact Medical Connections.

There is a potential problem with DicomImages’ ReadFile, ReadStream and ReadField methods in Borland languages, as these methods return a reference to a DicomImage or DicomDataSet object which is often not used (the item placed in the collection being used instead). In most languages, this extra reference is released immediately, but it seems that Borland languages create a hidden variable, which is not released until the current function ends. This can cause problems with file deletion etc., so if it is necessary to ensure that all references to an image (or dataset) are released, then a line such as:

```
images.Readfile(filename);
```

Should be replaced by:

```
image:=images.Readfile (filename);
image:=nil;
```

(equivalents apply in C++, using NULL)

### **15.5.1 Error Handling**

Error handling for ActiveX generated errors appears to be seriously lacking in Borland C++ Builder, and though Borland have promised improvements in the future it is currently almost impossible to trap and handle errors within the program.

### **15.6 Java**

DicomObjects is not a native Java collection, and there is currently no Java bean version. However, provided you are using only Windows, there are several Java ⇔ ActiveX/COM bridges commercially available, and DicomObjects has successfully been used with many of these. Note that the various bridges may differ greatly in their efficiency, and whilst most DicomObjects programmers make relatively few calls through the ActiveX interface, you should check the speed of various bridges if you have an application which will be making thousands of calls.

### **15.7 Other Environments**

There are now dozens of environments which support ActiveX/COM, and it would be impossible to cover every one. Most will support the majority of DicomObjects functionality with no problems at all, but for a few things you may need to be inventive. In particular, if your environment only supports the variant variable type, then make sure that potentially multi-valued (array) attributes are accessed using VariantValue rather than the default Value property.

## 16 Logging

DICOM is a complex, relatively new protocol, and there are very many possibilities for errors to be made by yourself, by established implementations or (even!) by DicomObjects, so when operations fail, either during development, or after installation at a user site, it is important to be able to obtain enough information to diagnose, and where possible, fix or circumvent the problem. To provide this information, DicomObjects has extensive log facilities, which can be accessed and controlled in several ways. The data available is the same, whichever access method is used, so this will be considered first, followed by access details.

### 16.1 Log Details and Levels

DicomObjects allows quite fine control over which information to log, using a bit field with the following values:

Decimal	Hex	Information Logged
1	1	Errors
2	2	Warnings
4	4	Informational Messages
8	8	Detailed Logging
16	10	All DICOM attributes received or read
32	20	All DICOM attributes sent or written
64	40	Byte-level data received, except contents of data PDUs
128	80	Byte-level data sent, except contents of data PDUs
256	100	ALL bytes received
512	200	ALL bytes sent

The levels normally used are:

- 0x7: High level logging only
- 0x3F: Details of all attributes sent and received
- 0xFF: Byte-level data, excluding data
- 0x3FF: **Full** logging, including pixel data

In general, 0x7 is useful for continuous logging of servers etc., 0x3F for semantic problems (missing attributes, wrong values etc.), and 0xFF for communication errors. 0x3FF should be used with great caution, where images (as opposed to just queries etc.) are transferred, as the log can become huge, and program execution slows to a crawl. Where appropriate (i.e. excluding PDU/PDV values etc.) the same levels apply to file reading and writing, as to network usage.

### 16.2 File Logging

Recording to a file is the main method of logging in DicomObjects, as files can easily be e-mailed, printed and analysed. There are three methods of controlling such logging:

- **Direct manipulation of registry keys**

This method has the advantage that it can be applied to absolutely any DicomObjects based

program without needing to make any changes whatsoever to the program itself, though a small disadvantage is that any changes to the registry keys apply only when the program is restarted.

- **Manipulation of the registry cache using a DicomGlobal object**

This method requires support within the program, but is very flexible, and logging can be started, stopped or modified at will.

- **Methods of the DicomLog control**

This method, likewise allows fine control, but requires a DicomLog object.

The registry keys involved, which may be modified either in the real registry or via DicomGlobal.RegWord are:

- **Log (DWORD)**

This is either 0 or 1. 1 enables logging, 0 (default) disables.

- **Loglevel (DWORD)**

This controls the level of detail as above.

- **LogLocation (String)**

The directory into which log files are written. They are always called D.O.xxxx.log, where xxxx represents the date and time. If absent, the logs are put in C:\.

An advantage of this method when investigating crashes is that every single entry is flushed to disk as soon as it is made, so the record is complete right up to the point of the crash, but a downside of this is that logging may slow the whole process and/or machine considerably, especially if all pixel data is logged at byte level! The logging may, under these circumstances, cause sufficient slowing that other applications timeout on network connections. The log files created are opened using a mode which allows read-access even while the log is being written.

### **16.3 The DicomLog Control**

This is the only visual control in DicomObjects other than the DicomViewer, and it receives exactly the same log information as file logging, but it has three extra facilities:

- The log messages can be displayed as they are produced.
- The messages are available to the program so programmers may do their own custom filtering if they wish.
- Logging to specific filenames may be controlled using the FileOpen method.

Please consult the help file for more details.

## 17 Advanced Usage

### 17.1 Over-riding Registry Values

Many aspects of DicomObjects are controlled by values held in the registry, which has the advantage that they can be controlled and changed without requiring changes to the program that uses them, but there are times when a program may need to know what values are in use, and also to be able to change them. For this reason, from version 4 on, DicomObjects implements a cache of the registry values, reading from the “real” registry only the first time that a value is accessed. As well as giving speed improvements, this enables the cache to be modified directly, and this is done using the RegWord and RegString indexed properties of a DicomGlobal object. These properties are read-write, and any changes apply immediately.

The root of all DicomObjects registry keys is:

```
HKEY_LOCAL_MACHINE/Software/Medical Connections/DicomObjects
```

Values directly under this key are indexed as single names, e.g.

```
RegString("LogLocation")
```

Values within subkeys are similarly accessible using the \ character to delineate keys, and a trailing \ indicates the default value of a key. See RegWord/RegString in the help file for more details.

### 17.2 Altering the List of Default SOP Classes

There are two situations where DicomObjects needs to guess in advance which SOP classes/abstract syntaxes it will need for an association:

- When initiating a C-GET operation
- When using an outgoing DicomConnection  
(where items have not been added to the Syntaxes collection)

In both these cases, DicomObjects uses the space-separated list of SOP classes held in the AbstractSyntax registry value, which may be modified either directly, or using DicomGlobal.RegString, as in section 17.1 above. If this value is missing, the default list is:

- 1.2.840.10008.1.1
- 1.2.840.10008.5.1.4.1.1.1
- 1.2.840.10008.5.1.4.1.1.1.1
- 1.2.840.10008.5.1.4.1.1.1.1.1
- 1.2.840.10008.5.1.4.1.1.1.2
- 1.2.840.10008.5.1.4.1.1.1.2.1
- 1.2.840.10008.5.1.4.1.1.1.3
- 1.2.840.10008.5.1.4.1.1.1.3.1
- 1.2.840.10008.5.1.4.1.1.2
- 1.2.840.10008.5.1.4.1.1.3
- 1.2.840.10008.5.1.4.1.1.3.1
- 1.2.840.10008.5.1.4.1.1.4
- 1.2.840.10008.5.1.4.1.1.6
- 1.2.840.10008.5.1.4.1.1.6.1
- 1.2.840.10008.5.1.4.1.1.7
- 1.2.840.10008.5.1.4.1.1.11.1
- 1.2.840.10008.5.1.4.1.1.12.1
- 1.2.840.10008.5.1.4.1.1.12.2
- 1.2.840.10008.5.1.4.1.1.20

- 1.2.840.10008.5.1.4.1.1.77.1.1
- 1.2.840.10008.5.1.4.1.1.77.1.2
- 1.2.840.10008.5.1.4.1.1.77.1.3
- 1.2.840.10008.5.1.4.1.1.77.1.4
- 1.2.840.10008.5.1.4.1.1.128

In future versions of DicomObjects, this list may be expanded, but all current SOP classes should be retained.

### 17.3 Transfer Syntax Selection

In most cases, the default transfer syntaxes proposed and accepted by DicomObjects for network operations will be appropriate for your needs, but there are cases where fine control may be necessary, especially for teleradiology applications and others operating over slow links. There are 3 basic methods for controlling the transfer syntaxes negotiated:

#### 17.3.1 TransferSyntax Registry Key (SCU & SCP)

When DicomObjects needs to find a default list of transfer syntaxes to propose, or to choose which to accept, it looks for a value with the name of the SOP class (abstract syntax) involved, within the following key:

`HKEY_LOCAL_MACHINE/Software/Medical Connections/DicomObjects/TransferSyntax`

If no such value exists, the default value of the above key itself is used, and if this is absent, an internal default list is used, which is:

- 1.2.840.10008.1.2.1 (EXPLICIT\_VR\_LE)
- 1.2.840.10008.1.2 (IMPLICIT\_VR\_LE)
- 1.2.840.10008.1.2.4.57 (EXPLICIT\_VR\_LE\_LJPEG57)
- 1.2.840.10008.1.2.4.70 (EXPLICIT\_VR\_LE\_LJPEG70)
- 1.2.840.10008.1.2.5 (EXPLICIT\_VR\_LE\_RLE)
- 1.2.840.10008.1.2.4.51 (EXPLICIT\_VR\_LE\_JPEG51)

Instead of using the “real” registry, values may be manipulated by using `DicomGlobal.RegString`.

Whatever the source, the list should be a space separated list of transfer syntax UIDs. For outgoing associations, all are proposed, and for incoming associations, the first in the list matching any of those proposed is accepted. Therefore, the list should contain not only all UIDs you wish to allow, but they should be in decreasing order of preference. For DICOM compliance, the list should normally contain implicit VR little endian (1.2.840.10008.1.2), but explicit VR syntaxes should appear first, to avoid the known problems with implicit VR usage.

For outgoing use, any transfer syntax UIDs immediately preceded by an asterisk are proposed in their own presentation contexts, with all others grouped in a separate presentation context. This allows the SCP an opportunity to accept more than one. The asterisks are ignored when accepting an association.

This is the only form of control allowed for the simple `DicomImage.Send` method, but for other operations, alternatives exist as below:

#### 17.3.2 Queries (SCU)

A `DicomQuery` object has a `TransferSyntaxes` property, which, if non-blank, over-rides the default registry-based value for any associations created by that object. The format is exactly the same as described above, and it is also used to select from incoming proposed transfer syntaxes while accepting secondary incoming associations as part of `GetUsingMove`.

### 17.3.3 DicomConnection (SCU only)

Before calling SetDestination (which actually negotiates an association), an application may modify the DicomConnection's Contexts property, and normally, this consists of just a series of Add methods to include all the required SOP classes. However, once a SOP class has been added, the associated transfer syntaxes may also be specified by modifying the DicomContext item's OfferedTS property, which may be set to either a single string value, or an array of strings. e.g.

```
Set connection=new DicomConnection
Connection.Contexts.Add 1.2.840.10008.5.1.4.1.1.7
Connection.Contexts(1).OfferedTS="1.2.840.10008.1.2.1"
Connection.Contexts.Add 1.2.840.10008.5.1.4.1.1.2
Connection.Contexts(3).OfferedTS=Array("1.2.840.10008.1.2",
    ↵"1.2.840.10008.1.2.4.51")
```

This code proposes secondary capture SOP class, with just explicit VR little endian, and CT, with both implicit VR little endian, and lossy JPEG.

Note that indexing of a DicomContexts collection is by presentation context ID, which must always be odd, so the indices will be 1,3,5 etc., not 1,2,3.

If OfferedTS is left blank, then the default registry values will be used.

### 17.3.4 In AssociationRequest (SCP)

By iterating through the proposed contexts and looking at the values in the OfferedTS array, an application may select which one it wishes to use, and then assign that to the AcceptedTS property. If this property is not assigned, then the default acceptance policy, using registry values is applied.

## 17.4 Private SOP classes

As DicomObjects is non-SOP class specific, you are free if you wish to define your own private and extended standard SOP classes. If you wish these to be displayed by DicomObjects, then the usage of pixel related attributes (mainly the 0x0028 group) should be standard.

## 17.5 Private Transfer Syntaxes

Custom DLLs can be used to implement private transfer syntaxes, enabling custom compression methods to be used. Full details may be found in the help file under "Compression and Decompression DLLs".

## 17.6 Storage Commitment

Storage commitment is a relatively new facility in DICOM, whereby a server explicitly takes responsibility for the long-term storage of an image, this being regarded as a separate step from the simple receipt via C-STORE, with a status of 0 (success). DicomObjects provides all the tools needed to implement storage commitment as either an SCU or SCP, these being:

- N-ACTION support (SCU & SCP)
- N-EVENT-REPORT support (SCU & SCP)
- Reverse rôle negotiation (SCU initiates association)

These procedures require a detailed knowledge of the underlying DICOM protocols, and in order to implement this facility, you will need to study the standard closely, but the following slightly unusual aspects of DicomObjects will need to be considered:

- In order to implement reverse role negotiation, the default values of RequestorSCURole & RequestorSCPRole (properties of a DicomContext) will need modifying, either before calling DicomConnection.SetDestination (when initiating as SCP), or in AssociationRequest (when

accepting as SCU). Clearly, these should be applied only to presentation contexts using the storage commitment SOP class.

- NEventReport events are fired differently according to the mode of the DicomConnection. In order to be able to respond to them with a non-zero status (if you wish to reject them), the DicomConnection's Mode property must be doNoSync.

## **18 DicomObjects for Windows CE**

As of version 4.1, DicomObjects is available for the Windows CE environment, allowing use on devices such as PDAs or some tablets. Due to the range of devices and processors, such versions are made only on request, but this can be done easily. The vast majority of the functionality of DicomObjects is identical under CE, but a few limitations exist due to the environment:

- DirectDraw is not used
- Picture, Paste and WriteAVI methods are not available
- Circular and polygonal shutters are not applied
- Memory mapping is not used for image files.

In addition, the following limitations apply to DicomLabel objects:

- All text is left aligned (Alignment property of DicomLabel objects is ignored)
- Arc labels are not drawn, and interpolated labels are drawn as polygons
- BoundingPolygon and all ROI functions are not supported
- Labels do not rotate (Angle is ignored)