A Portable Toolkit for Providing Straightforward Access to Medical Image Data[1]

Robert J.T. Sadleir, BEng

Paul F. Whelan, PhD

Padraic MacMathuna, MD

Helen M. Fenlon, MB

This work was supported by the Irish Cancer Society, the Health Research Board (of Ireland) and the Science Foundation Ireland.

Address correspondence to R.J.T.S.
Vision Systems Laboratory,
School of Electronic Engineering
Dublin City University
Dublin 9,
Ireland.
Tel: +353 1 7008592
Fax: +353 1 7005508
E-mail: Robert.Sadleir@dcu.ie

Exhibit Space Number: 9316

[1] From the Vision Systems Group, School of Electronic Engineering, Dublin City University, Ireland. (R.J.T.S., P.F.W.); the Gastrointestinal Unit, Mater Misercordiae Hospital, Dublin 7, Ireland. (P.MacM); and the Department of Radiology, Mater Misercordiae Hospital, Dublin 7, Ireland. (H.M.F.);

A Portable Toolkit for Providing Straightforward Access to Medical Image Data

Abstract

Computer aided medical image analysis is emerging as an important area in radiology research. Analysis of medical images usually involves the development of custom software applications that interpret, process and ultimately display medical image data. The interpretation stage is generally the same for all such applications and involves decoding image data and presenting it to the application developer for further processing. This article introduces a toolkit created specifically for interpreting medical image data and thus acting as a platform for medical imaging application development. The toolkit, which will be referred to as NeatMed, is intended to reduce development time by removing the need for the application developer to deal directly with medical image data. NeatMed was implemented using Java, a programming language with a range of attractive features including portability, ease of use and extensive support material. NeatMed was developed specifically for use in a research environment. It is straightforward to use, well documented and is intended as an alternative to commercially available medical imaging toolkits. NeatMed currently provides support for both the digital imaging and communications in medicine (DICOM) and Analyze medical image file formats and its development is ongoing. Support material including sample source code is available via the Internet and links to related resources are also provided. Most importantly NeatMed is freely available and its continuing development is motivated by requests and suggestions received from end users.

One sentence summary: This article introduces a freely available and user friendly Java toolkit that is intended to act as a platform for the development of custom medical imaging applications.

Introduction

Custom medical imaging applications are becoming more common place. This is primarily driven by the increasing role of computer aided image processing and analysis in radiology research. An import and common stage in the development of such applications is the interpretation of medical image data. This data is generally stored in accordance with the Digital Imaging and Communications in Medicine (DICOM) standard (1) (summarised by Mildenberger et al. (2)). Interpretation of medical images involves decoding the relevant DICOM data and making it readily available to the application developer for analysis and display. Our group has developed a Java based medical imaging toolkit to facilitate the rapid development and deployment of medical imaging applications in a research environment. Over the past number of years this toolkit has been used internally as a platform for the development of a number of research applications in the areas of CT colonography (3, 4), MRCP (5) and more recently functional MRI. In the course of this work the toolkit has been exposed to a wide range of medical images obtained from various imaging modalities.

This medical imaging toolkit, which will be referred to as the NeatMed application developers interface (API), was developed using the Java programming language (Sun Microsystems, Mountain View, Calif.). The use of Java has previously been demonstrated in the development of custom medical imaging applications. Fernandez-Bayo et al. describe a web based Java viewer developed for use with a custom web server (6), Bernarding et al. describe a framework for the development of DICOM server applications implemented using Java (7) and Mikolajczyk et al. describe a Java environment for the analysis of positron emission tomography (PET) image data (8).

The choice of Java for implementing the NeatMed API was also influenced by a number of its key features:

- **Ease of use:** Java is modern programming language that was designed with simplicity in mind. Many of the complexities that are associated with other programming languages have been omitted while much of the power and flexibility has been retained. This makes Java very easy to learn and use particularly in the case of novice programmers.
- **Level of support:** Although Java is a relatively new programming language, there is a significant amount of support material available. Numerous texts have been written dealing with all aspects of the language. In addition: tutorials, sample source code, API documentation and freely available integrated development environments (IDEs) can all be accessed via the Internet.
- **Portability:** Java is a multi-platform programming language. This means that a Java application developed on one operating system (e.g. Mac OS) can be deployed on a number of other different operating systems (e.g. Windows, Solaris and Linux). This is sometimes referred to as the "write once, run anywhere" paradigm. In theory, the use of a multi-platform programming language significantly increases the potential user base with little or no extra development overhead.
- **Core functionality:** The core Java libraries encapsulate an extensive range of functionality that can be easily reused to create reliable, diverse and powerful applications. Some of the main capabilities supported by the core libraries include: networking, file IO, image processing, database access and graphical user interface development.

- **Extension APIs & toolkits:** There are a large number of extension APIs and toolkits that can be used in conjunction with the core Java libraries. These extensions usually deal with a particular speciality such as advanced image processing, 3D graphics or speech recognition.

The concept of a Java based medical imaging toolkit is not entirely new. In fact there are a number of commercial toolkits already available: the DICOM Image I/O Plugin (Apteryx, Issy-les-Moulineaux, France), the LEADTOOLS Medical Imaging suite (LEAD Technologies, Charlotte, N.C.) and the Java DICOM toolkit (Softlink, Halle Zoersel, Belgium). NeatMed was developed as a freely available alternative to these commercial toolkits. It is primarily intended for use in a research environment, specifically for dealing with offline medical image data. A major benefit associated with NeatMed is that it was designed with simplicity and ease of use in mind. This is especially important as it makes NeatMed assessable to all those involved in medical imaging research including radiologists, computer scientists and engineers. Other freely available toolkits e.g. ImageJ (9) and jViewbox (10) facilitate the development of applications that deal with DICOM images however they lack the low level functionality required to create application that deal with all aspects of the DICOM file format.

The NeatMed API was released prior to the 88th scientific assembly and annual meeting of the Radiological Society of North America. This article is intended to provide an introduction to the NeatMed API and deals with implementation issues, toolkit structure, sample applications, support material and current API status. The reader is encouraged to visit the official NeatMed website (9) for further information

about the NeatMed API and associated resources. The NeatMed API is distributed in accordance with the terms and conditions laid out in the GNU Lesser General Public License. This licence was select to insure that the NeatMed API is accessible to all potential users.

API Implementation

NeatMed was developed using the Java programming language. Java was created by Sun Microsystems in the early ‘90s. It was initially intended for the development of software for use in the consumer electronics industry (e.g. set-top boxes). Unfortunately the Java development team were unable to find a target market for their new programming language. Forced to reconsider, it was decided to redeploy Java as an Internet technology. Java programs known as Applets first appeared on the Internet in 1994. These interactive Applets were embedded in standard hypertext mark-up language (HTML) web pages and greatly enhanced the previously static Internet. The Java programming language can also be used for the development standalone client side applications which can run independently of a web browser and are not subjected to the same security restrictions as Applets. The core Java libraries maintained by Sun Microsystems are used as the foundation for the development of any Java application. These libraries can be used in conjunction with extension an API in order to develop specialised applications. See (11) for further information about the Java programming language.

An extension API is a set of classes that can be instantiated by a programmer to create a particular type of application, thus facilitating software reuse. NeatMed is an

example of an extension API that can be used for the development of applications that deal with offline medical image data. A large number of Java APIs exist, these deal with a broad range of applications ranging from communicating with the serial and parallel ports to advanced image processing. The set of classes representing the API is deployed in some type of library. Java provides a packaging tool that can be used to package a set of class files and associated resources into a Java archive or JAR file. In order to be useful an API must be well documented. Java provides a documentation tool called Javadoc that allows an API developer to document software as it is being written. The resulting documentation provides detailed information about each class, method and variable that is defined in the associated API. The structure of Javadoc documentation is more or less the same for every API. This makes it very easy for programmers to familiarise themselves with a new API once they are comfortable with the basic Javadoc documentation structure. The API documentation is generated in HTML and can be viewed using any standard web browser.

Java has a wide range of benefits associated with it however there are also some limitations. One example is performance: Java is a multiplatform programming language, the byte code (i.e. binary form) that represents a java program is interpreted and not executed directly. This reduces the performance of a Java program compared to a natively executed program. Overall however, the benefits associated with Java (listed previously) far outweigh the drawbacks hence its selection for the development of NeatMed.

API Overview

The NeatMed API is a group of core and support classes that can be used to interpret, represent and manipulate images and related data that are stored in DICOM compliant files. The central class in the API is the **`DICOMImage`** class. A **`DICOMImage`** object can be instantiated by specifying a reference to a suitable data source in the constructor. The constructor will accept data from a number of sources (e.g. local file, data stream and remote URL). Once constructed a **`DICOMImage`** object provides direct access to all of the data elements stored within the specified DICOM source. Other classes in the API are used to represent individual components within a DICOM file such as **`DICOMTag`**: a data element tag (i.e. group number and element number combination) or **`DICOMDataElement`**: a special wrapper class created for storing data associated with a particular value representation. There is a subclass of **`DICOMDataElement`** for each of the 26 value representations, examples include: **`DICOMAgeString`**, **`DICOMPersonName`** and **`DICOMUniqueIdentifier`** The API also provides utilities classes such as **`DICOMDataDictionary`**: a data dictionary containing value representation definitions for all of the data elements defined in the DICOM 3.0 standard (required when the value representation is not specified explicitly). This section describes how the **`DICOMImage`** class operates and how it interacts with other classes in the API to provide access to DICOM encoded data.

Data Interpretation

When a **`DICOMImage`** object is constructed it reads and decodes all the data from the specified source. The decoder automatically determines the type of data and the transfer syntax to be used. If the value representation is not specified explicitly then

the required value representation for a particular data element is obtained from the pre-programmed data dictionary (i.e. **`DICOMDataDictionary`**). Each data element in the DICOM file is subsequently decoded and stored using the relevant wrapper class. Data stored within a wrapper class can easily be accessed for further processing. The image data **`(0x7fe0, 0x0010)`** is a special type of data element that can be stored using one of two possible value representations: other byte string or other word string. The image data is always packed and sometimes compressed using either joint photographic experts group (JPEG) (13, 14) or run length encoding (RLE) compression. Information required to unpack and decompress the image data is usually stored within group **`0x0028`** data elements. The DICOM decoder automatically unpacks and decodes the image data and stores it internally as a one dimensional array of signed 32-bit integer primitives.

Data Management

Once decoded all data elements are stored internally using a modified hash table structure. A hash table allows information to be stored as key/value pairs. In this case the key is the data element tag which consists of a group number and an element number and the value is the data element associated with the relevant tag. A tag is represented by a **`DICOMTag`** class and all data elements are represented by dedicated wrapper classes which are subclasses of **`DICOMDataElement`**. A specific data element value can be retrieved from the hash table by specifying the relevant group number and element number combination. The method employed for storing and querying decoded data elements is illustrated in Figure 1.

Data Access/Manipulation

Data can be accessed at several levels. At the most basic level a data element can be retrieved from the hash table using the **`getDataElement()`** method of the **`DICOMImage`** class. There are two versions of the **`getDataElement()`** method. The first version takes a single argument of type **`DICOMTag`** which encapsulates the group number and element number of the desired data element and the second version takes two integer arguments, one for the group number and the second for the element number. In either case the **`getDataElement()`** method will return an object of type **`DICOMDataElement`** representing the requested data element. This object must be cast (i.e. converted) into the relevant subclass of **`DICOMDataElement`** before the actual data element value can be accessed. An existing data element can be modified or a new data element can be added to a **`DICOMImage`** object by calling the **`setDataElement()`** method. This method takes arguments that represent a tag and the new data element value to be associated with that tag. The basic level representation does not provide direct access to the actual image data. This data is packed and in some cases compressed and further decoding is required in order to facilitate pixel level operations.

There are an extremely large number of data elements defined by the DICOM standard. It is possible for any of these data elements to be present in a DICOM file. Some data elements occur more frequently than other and have particular significance to the application developer. In order to facilitate straightforward access to important and frequently used data elements a number of special accessor methods are defined by the **`DICOMImage`** class. This set of methods represents a higher level of access than that provided by the **`getDataElement()`** and **`setDataElement()`** methods and simplifies access to particular frequently used or important data

elements. Internally each accessor method calls the **`getDataElement()`** method with the relevant **`DICOMKey`** argument and casts the returned data element value into the relevant Java primitive (or object). Some examples of the accessor methods provided by the **`DICOMImage`** class are as follows:

- **`String getPatientID()`** retrieves the data element value associated with the key **`(0x0010, 0x0020)`** from the hash table. The **`DICOMLongString`** object at this location is converted into a Java **`String`** object which is then returned.
- **`int getSeriesNumber()`** retrieves the data element value associated with the key **`(0x0020, 0x0011)`** from the hash table. The **`DICOMIntegerString`** object at this location is converted into a signed 32-bit integer primitive and returned.
- **`int getBitsAllocated()`** retrieves the data element value associated with the key **`(0x0028, 0x0100)`** from the hash table. The **`DICOMUnsignedShort`** object at this location is converted into a signed 32-bit integer primitive and returned.

Accessor methods are also provided for adding new data element values or modifying existing data element values. These methods take a single argument which represents the new value of the relevant data element. Internally these methods call **`setDataElement()`** with the relevant **`DICOMKey`** and new value arguments. Examples include **`setPatientID(String ID)`**, **`setSeriesNumber(int seriesNumber)`** and **`setBitsAllocated(int bitsAllocated)`**.

The final level of abstraction is used to provide direct access to image data. As mentioned previously the image data is stored in data element **`(0x7fe0,`**

0x0010). After the initial decoding/interpretation stage this data is still packed and sometimes compressed. The final stage of decoding performed by the **DICOMImage** class automatically unpacks and decompresses the image data which is ultimately stored within the **DICOMImage** class as a one dimensional array of signed integer values. Individual pixel values can be accessed/modified using special accessor methods: **getSample()** and **setSample()**. In either case the (x, y) coordinates of the relevant pixel must be specified in the argument list of the accessor method. In the case of multi-frame image data (i.e. where the number of frames is greater than one) a time coordinate must also be specified and for multi-plane image data (e.g. ARGB or CYMK) a colour plane index must also be specified. Some examples of the pixel level accessor methods provided by the **DICOMImage** class are as follows:

- **getSample(int x, int y)** returns an integer primitive that represents the pixel value at the specified (x, y) coordinates.
- **getSample(int x, int y, int t)** returns an integer primitive that represents the pixel value at the specified (x, y, t) coordinates.
- **setSample(int x, int y, int value)** sets the value of the pixel at the specified (x, y) coordinates to be the same as that indicated by the **value** argument.

Data Storage

In many cases an application developer will only need to read from a DICOM file. However, there are some cases where it may be useful to modify the contents of a DICOM file. In order to facilitate this operation the **DICOMImage** class provides a **save()** method. This method saves the current state of the associated **DICOMImage** object and takes a single argument which represents the output stream where the data

will be written. The saving operation involves writing all of the data elements represented by the **`DICOMImage`** object (including the packed pixel information) to the specified output stream. The saved data will be stored according to the specified transfer syntax, i.e. data element **`(0x0002, 0x0010)`**. If this data element is not present then the default transfer syntax will be used. The data storage feature completes the range of functionality provided by the NeatMed API.

Sample Applications

The NeatMed API can be used to develop of a wide variety of medical imaging applications. This section describes a number of sample applications that demonstrate the power, flexibility and ease of use of the NeatMed API (see Figure 3). In some cases NeatMed is used in conjunction with extension APIs and toolkits in order to demonstrate its development potential. The source code and associated instructions for the compilation and execution of each sample application can be downloaded directly from the NeatMed website.

Simple DICOM viewer

A simple DICOM image viewer application can be created quite easily. The source code for the application is extremely compact (see Figure 3). The application takes a single argument which represents the name of the image to be displayed. This filename is subsequently used to construct a **`DICOMImage`** object. A **`BufferedImage`** object representation of the **`DICOMImage`** object is then obtained. This **`BufferedImage`** object is ultimately displayed using Swing graphical user interface (GUI) components (see **`DICOMViewer.java`**).

Sequence Viewer

The simple DICOM viewer application can be extended to deal with multi-frame DICOM images. This extension is facilitated by adding two buttons to the application GUI. These buttons allow the user to sequence backwards and forwards through the available frames. Internally the buttons update the value of a counter, the backwards button decrements the counter and the forwards button increments the counter. After either button is pressed the frame with the index that corresponds to the counter value displayed. This application demonstrates how user interaction is handled by Java. (see **`SequenceViewer.java`**).

Volume Rendering

The NeatMed API can be used in conjunction with the visualisation toolkit (VTK) (15) to provide three dimensional volume or surface renderings of medical image data. NeatMed is used to read in axial slices from a volumetric dataset. The image data from the slices is used to populate an array of scalar values that represents the volumetric data. Two transfer functions (opacity and colour) are specified to indicate how the volume should be displayed. The opacity transfer function indicates the opacity values associated with particular voxel intensity. The colour transfer function indicates the RGB colour value associated with a particular voxel intensity, this can be used for pseudo colouring the volume data. The rendering of the volume is handled by the VTK and the user can interact with the resulting model using the mouse. Supported operations include rotate, zoom and translate. A sample volume rendering using this application is illustrated in Figure 4 (see **`VolumeRendering.java`**).

Anonymising Data

A DICOM file usually contains a number of data elements that hold sensitive patient information. It is important to be able to anonymise this information in certain cases in order to protect the patient's identity. The anonymisation process involves constructing a **`DICOMImage`** object from a suitable source, overwriting all of the sensitive data element values (e.g. patient's name, patient's birth date and patient's address) and saving the modified **`DICOMImage`** object to an output stream using the **`save()`** method. The saved file represents the anonymized data. Each of the sensitive data element values can be overwritten using the **`setDataElement()`** method or special accessor methods (where available). Note that this application is a command line only application there is no GUI (see **`Anonymise.java`**).

Image Processing

The NeatMed API can be used for the development of medical image analysis applications. This can be achieved using any of the well documented image processing operations that are described in the literature, see (11) for examples of image processing algorithms in Java. The threshold operation is a simple example of a point operation that is based on a raster scan of the input image. Thresholding converts a grey scale image into a binary (black & white) image based on a user defined threshold value. Any pixel that is greater than or equal to the threshold is set to white while any pixel less than the threshold is set to black. The threshold operation performs a rough segmentation of the image and demonstrates both looping and conditional data processing. The threshold operation is illustrated in Figure 5 (see **`Threshold.java`**).

As mentioned earlier, NeatMed has been used internally over the past number of years as a platform for the development of a number of medical imaging research applications. Examples of this research are illustrated in Figure 6.

Support Material

All support material for the NeatMed API is accessible through the official NeatMed website. In addition to providing access to the latest version of the NeatMed API the website also provides access to documentation, sample source code and contact information for the NeatMed development team.

The NeatMed API documentation is generated using Javadoc. The NeatMed API documentation (see Figure 7) can be browsed online or downloaded as a ZIP file for subsequent offline access. In either case the documentation can be viewed using a standard web browser. The documentation itself provides an overview of the entire API. Each class and its associated methods and variables are described in detail. The relationship between classes is indicated and hyperlinks are provided to facilitate easy navigation through the documentation.

Fully commented source code for each of the sample applications described earlier in this paper can be downloaded from the NeatMed website. These applications can be compiled and executed using the Java 2 Software Developer Kit (J2SDK) from Sun Microsystems. The latest version of the J2SDK (currently v1.4.2) can be downloaded at no cost from official Java website (16). All sample applications require the

NeatMed API to be included in the Java class path and some applications also require additional APIs or toolkits such as the VTK to be included. The NeatMed website should be consulted for detailed information about compiling and executing the sample applications. Links to sites providing DICOM images that can be used in conjunction with the sample applications can also be accessed via the NeatMed website.

Finally the NeatMed team can be contacted by email either for general queries or to suggest additional functionality that may be useful to other NeatMed users. All queries regarding the NeatMed API should be directed to neatmed@eeng.dcu.ie

Current Status

The NeatMed API is constantly evolving. Its ongoing development is driven by internal requirements and external requests for additional features. NeatMed supports data that conforms to version 3 of the DICOM standard as well as the older American College of Radiology – National Electrical Manufacturers Association (ACR-NEMA) standard. NeatMed currently supports all of the uncompressed DICOM transfer syntaxes as well as the lossless RLE transfer syntax (1.2.840.10008.1.2.5). Support for JPEG compressed image data is currently being incorporated into the API. Dedicated wrapper classes are provided for storing each of the 26 possible data element value representations defined in the DICOM standard. The pixel data decoder is extremely flexible and supports any valid pixel encoding (packing) scheme, the majority of commonly used photometric interpretations are also supported and the

recent addition of the basic data write feature completes the range of functionality available for dealing with DICOM data.

NeatMed has recently been updated to support version 7.5 of the ANALYZE file format. As with DICOM data, the ANALYZE file pair (image & header) is automatically interpreted and all encoded information is made readily available to the programmer. The ANALZE file format has a finite number of header fields and direct access is provided to each of these. Individual image sample values can be obtained by simply specifying the relevant 2D, 3D or 4D co-ordinates. The ANALYZE section of the API can also be used in conjunction with other toolkits and APIs to create powerful medical imaging applications.

The NeatMed API is intended for use by programmers interested in developing medical imaging applications, particularly for computer aided analysis. There is no reason why nonprogrammers should be excluded from developing such applications. In order to facilitate nonprogrammers NeatMed has recently been incorporated into the NeatVision visual programming environment (11, 17). NeatVision is a free, Java based software development environment designed specifically for use in image analysis applications. NeatVision is both user-friendly and powerful providing high-level access to over 300 image manipulation, processing and analysis algorithms. A visual program can be created by simply instantiating the required algorithms (blocks) and creating data paths (interconnections) between the inputs and outputs to indicate the data flow within the program. This combination NeatVision and NeatMed makes a vast library of image processing operators easily accessible to the medical image analysis community.

Summary

The interpretation of medical image data is an important and common stage in the development of any medical imaging application. NeatMed removes the need to deal directly with encoded medical image data, thus increasing productivity and allowing the developer to concentrate on other aspects of application development. NeatMed is written in Java, a multiplatform programming language with a large amount of freely available support material that is straightforward to learn and use. These and other features of Java make NeatMed accessible to a large group of potential users. NeatMed is well supported with documentation and sample code available through the NeatMed website. Most importantly, NeatMed is a freely available research tool whose ongoing development is driven by the needs and requirements of its users.

Acknowledgements: The authors wish to thank members of the Department of Radiology and the Gastrointestinal Unit at the Mater Misericordiae Hospital, particularly Dr. John F. Bruzzi and Dr Alan C. Moss.

References

1. National Electrical Manufacturers Association. Digital Imaging and Communications in Medicine (DICOM). Rosslyn, Va: National Electrical Manufacturers Association, 2003; PS 3.1-2003–3.16-2003.
2. Mildenberger P, Eichelberg M, Martin E. Introduction to the DICOM standard. Eur Radiol 2002; 12:920-927.
3. Sadleir RJT, Whelan PF. Colon centreline calculation for CT colonography using optimised 3D topological thinning. In: Proceeding of the 1st International Symposium on 3D Data Processing Visualisation and Transmission, Padova, Italy, Jun 19 – 21, 2002; 800-803.
4. Sadleir RJT, Whelan PF, Bruzzi JF, Moss AC, Fenlon HM, MacMathuna P. A novel technique for identifying the individual regions of the human colon at CT colonography. In: Proceedings of the IEE seminar on Medical Applications of Signal Processing, London, UK, Oct 7, 2002; 8/1-8/5.
5. Robinson K, Whelan PF, Stack J. Segmentation of the biliary tree in MRCP data. In: Proceeding of OPTO-Ireland: SPIE's Regional Meeting on Optoelectronics, Photonics and Imaging. Galway, Ireland. Sept 5 – 6, 2002; 192-200.
6. Fernandez-Bayo J, Barbero O, Rubies C, Sentis M, Donoso L. Distributing medical images with internet technologies: a DICOM web server and a DICOM java viewer. Radiographics 2000; 20:581-591.
7. Bernarding J, Thiel A, Decker I, Tolxdorff T. Implementation of a dynamic platform-independent DICOM-server. Comput Methods Programs Biomed 2001; 65:71-78

8. Mikolajczyk K, Szabatin M, Rudnicki P, Grodzki M, Burger C. A Java environment for medical image data analysis: initial application for brain PET quantitation. Med Inform 1998; 23:207-214.
9. ImageJ, Image Processing and Analysis in Java. Available at *http://rsb.info.nih.gov/ij/index.html*. 2003; Accessed December 5.
10. JViewbox, Laboratory of Neuro Imaging, UCLA. Available at *http://www.loni.ucla.edu/Software/Software_Detail.jsp?software_id=1*. 2003; Accessed December 5.
11. Whelan P, Molloy D. Machine Vision Algorithms in Java: Techniques and Implementations. 2nd ed. London: Springer-Verlag, 2001.
12. NeatMed Medical Imaging Application Programmers Interface. Available at *http://www.eeng.dcu.ie/~vsl/DICOM/index.html*. 2003; Accessed December 5.
13. Wallace GK. The JPEG still image compression standard. Commun ACM 1991; 34:30-44.
14. Pennebaker WB, Mitchell JL. JPEG still image data compression standard. 1st ed. New York: Chapman & Hall, 1993.
15. Schroeder WJ, Avila LS, Hoffman W. Visualizing with VTK: A tutorial. IEEE Comput Graph 2000; 20:20-27
16. Sun Microsystems Inc. J2SE Downloads. Available at *http://java.sun.com/j2se/downloads.html*. 2003; Accessed December 5.
17. The NeatVision visual programming environment. Available at *http://www.neatvision.com/index.html*. 2003; Accessed December 5.

Illustrations

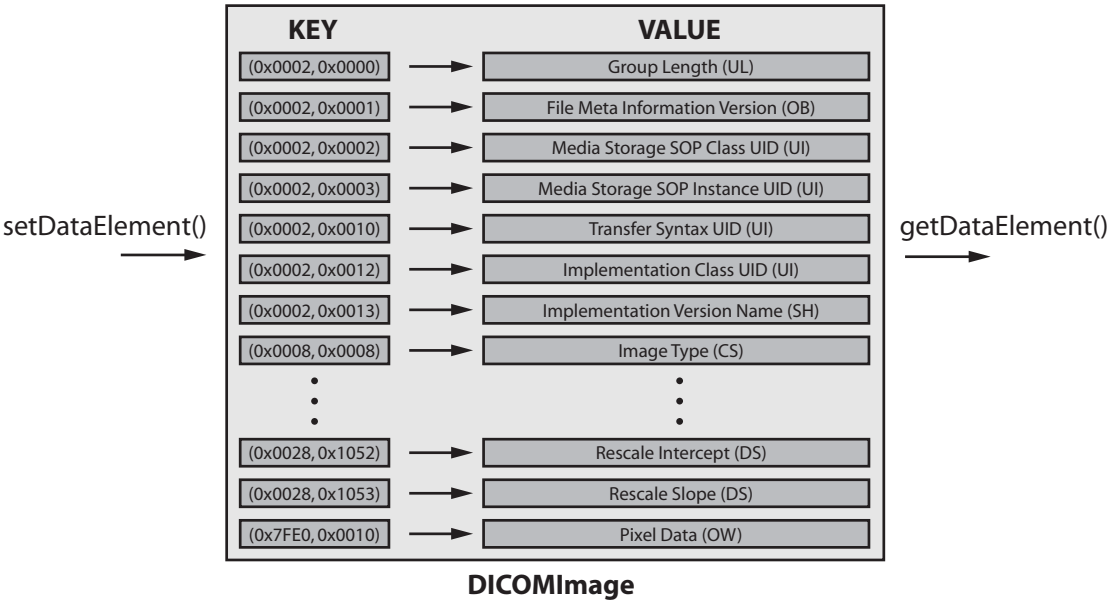


Figure 1: A representation of how data is stored and queried at the most basic level of the **`DICOMImage`** class.

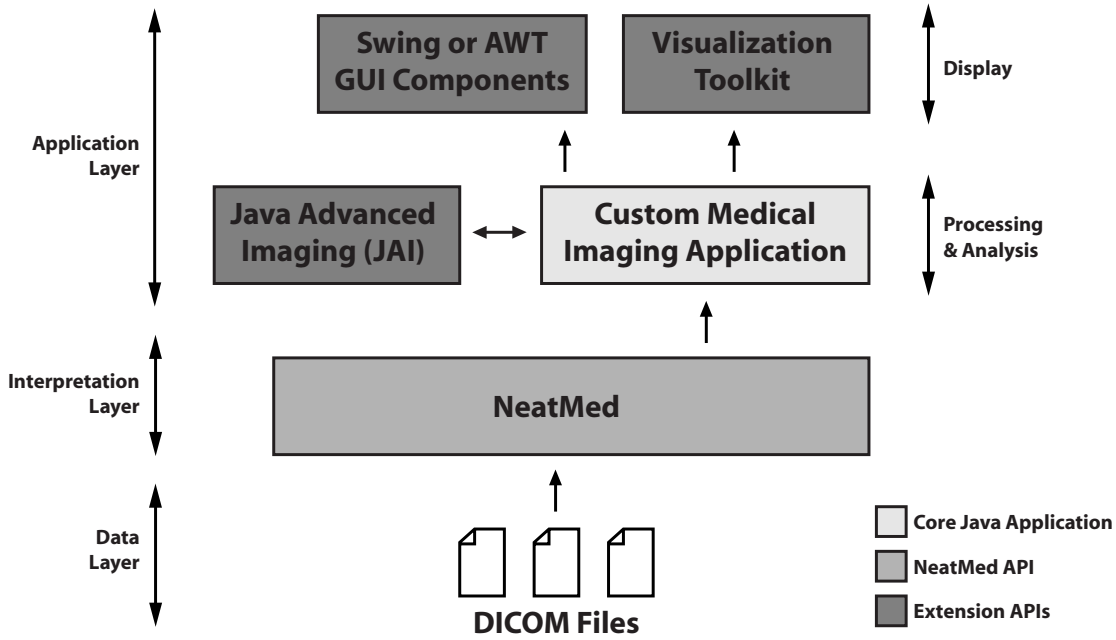


Figure 2: An overview of how the NeatMed API should be used in the development of medical imaging applications.

```
import java.awt.*;
import javax.swing.*;
import neatmed.DICOM.*;

public class ImageViewer
{
    public static void main(String[] args)
    {
        new ImageViewer(args);
    }

    public ImageViewer(String[] args)
    {
        try
        {
            // Create a new DICOMImage object...
            DICOMImage image = new DICOMImage(args[0]);

            // Create a GUI to display the image...
            JLabel label = new JLabel(new ImageIcon(image.getAsBufferedImage()));
            JFrame frame = new JFrame("DICOM Image Viewer");
            frame.getContentPane().setLayout(new BorderLayout());
            frame.getContentPane().add(label);
            frame.setVisible(true);
            frame.pack();
         }
         catch(Exception e)
         {
             System.out.println(e.toString());
         }
    }
}
```

Figure 3: Sample Java source code for the DICOM image viewer application.

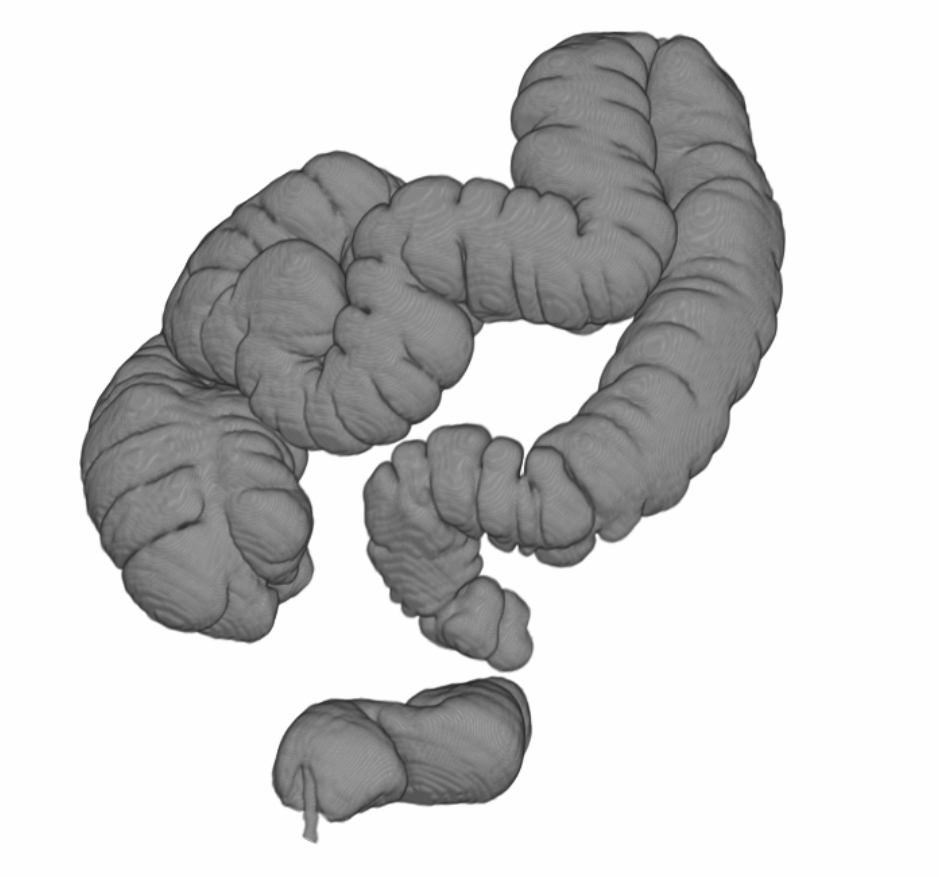
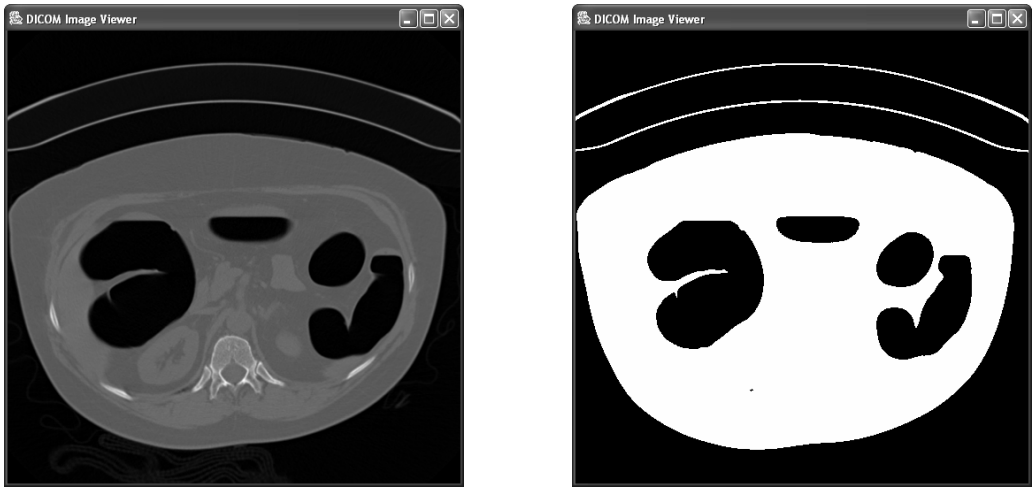
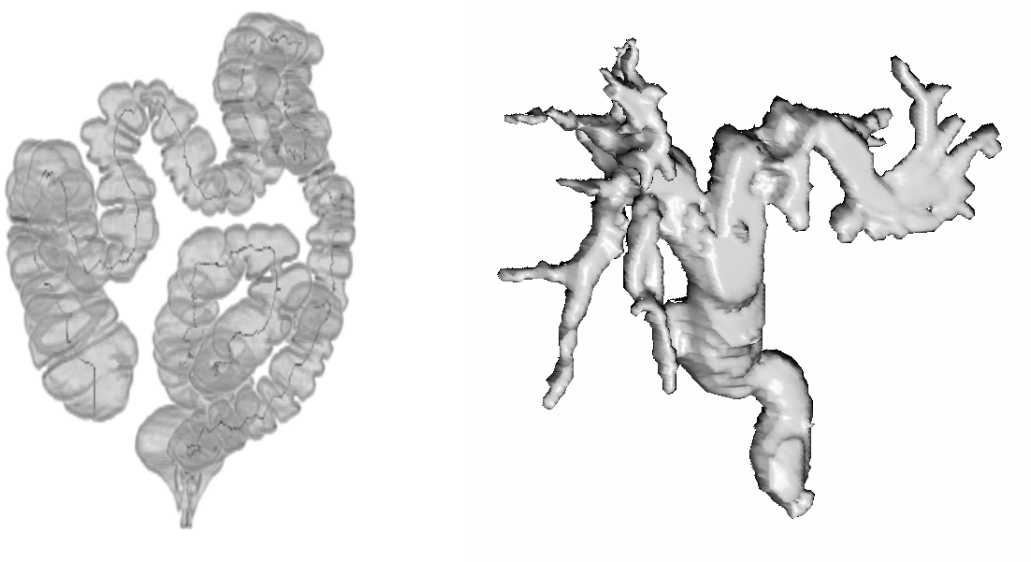


Figure 4: A volume rendering of the human colon generated from an abdominal CTC dataset using a combination of the NeatMed API and the VTK.



(a) (b)

Figure 5: The threshold image processing operation. A CT image (a) before and (b) after application of the threshold.



(a) (b)

Figure 6: Advanced image processing facilitated by NeatMed (a) Centreline calculation at CT colonography. (b) Segmentation of the biliary tree from MRCP data.

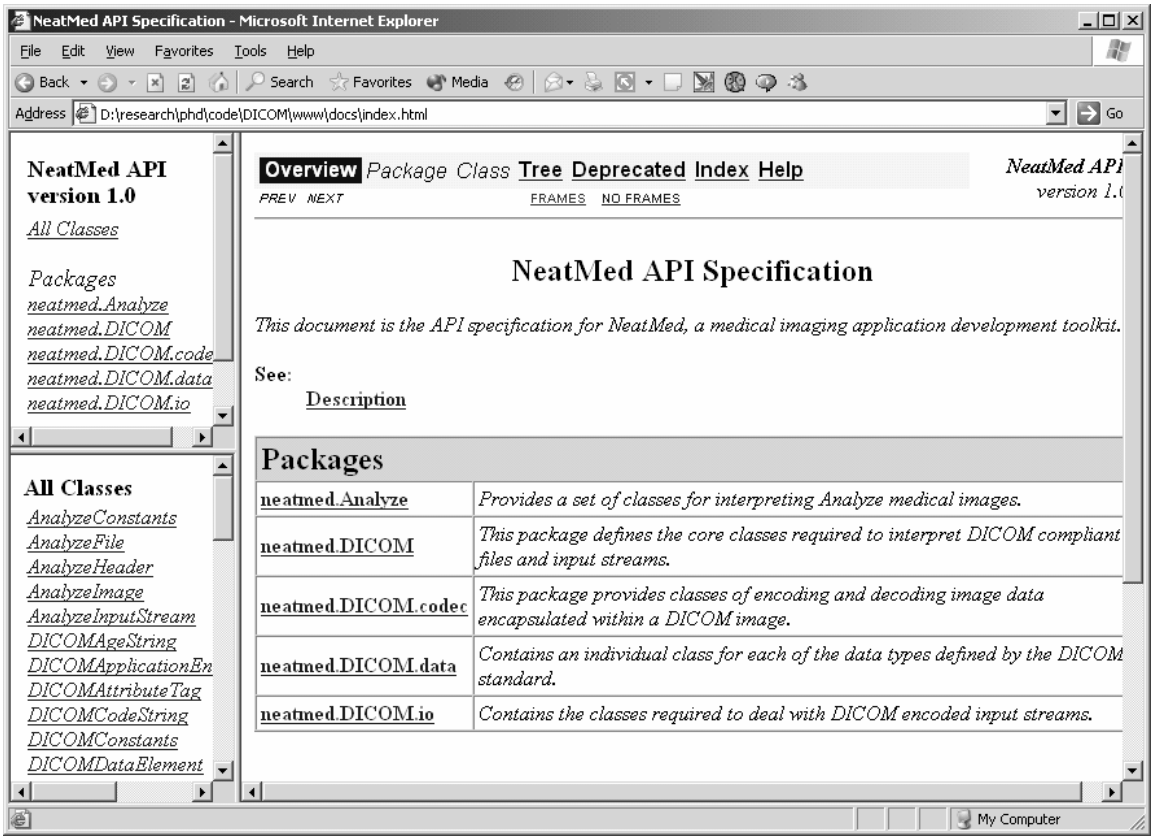


Figure 7: The HTML Javadoc documentation for the NeatMed API.