Merge DICOM Toolkit™

5.17.0

Python User's Manual

Merge Healthcare Incorporated
900 Walnut Ridge Drive
Hartland, WI 53029
USA

Symbols Glossary:

Symbol	Title
	Manufacturer
	Consult Instructions for Use

The full symbols glossary can be viewed at https://www.merative.com/content/dam/merative/documents/brief/Merge_Healthcare_Symbols_Glossary.pdf.

For application support or to report issues with user documentation, contact Customer Support:

1-877-741-5369 (North America)
+31.20.514.5073 (Europe, the Middle East and Africa)
1.800.952.156 (Australia)

MC3Support@merative.com

Part	Date	Revision	Description
COM-5488	July 2023	1.0	Updated bi-annually

The latest version of this document can be found at https://mergecustomer.force.com/mergeusercommunity/login.

Contents

Chapter 1. Overview

This User's Manual is intended for developers of medical imaging applications who are using the Merge DICOM Toolkit to provide DICOM network or media functionality.

The Merge DICOM Toolkit supplies you with a powerful and simplified interface to DICOM. It lets you focus on the important details of your application and the immediate needs of your end users, rather than the complex and often confusing details of the DICOM Standard.

The goal of this manual is to give you basic understanding of DICOM, and a clear understanding of the Merge DICOM Toolkit.

1.1. The DICOM Standard

The Digital Imaging and Communications in Medicine (DICOM) Standard was originally developed by a joint committee of the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) to, "facilitate the open exchange of information between digital imaging computer systems in medical environments." [1]

[1] NEMA Standards Publication No. PS 3.5-1993; DICOM Part 5 - Data Structures and Encoding, p.4.

Since its initial completion in 1993, the standard has taken hold. More and more products are advertising DICOM conformance, and more customers are requiring it. DICOM has also been incorporated as part of a developing European standard by CEN, as a Japanese standard by JIRA, and is increasingly becoming an international standard.

The DICOM Standard 2011 edition is composed of thousands of pages over 18 separate parts (parts 9 and 13 have been retired). Each part of the standard focuses on a different aspect of the DICOM protocol:

Part 1: Introduction and Overview

Part 2: Conformance

Part 3: Information Object Definitions

Part 4: Service Class Specifications

Part 5: Data Structures and Encoding

Part 6: Data Dictionary

Part 7: Message Exchange

Part 8: Network Communication Support for Message Exchange

Part 9: Point-to-Point Communication Support for Message Exchange (retired)

Part 10: Common Media Storage Functions for Data Interchange

Part 11: Media Storage Application Profiles

Part 12: Media Formats and Physical Media for Data Interchange

Part 13: Print Management Point-to-Point Communication Support (retired)

Part 14: Grayscale Standard Display Function

Part 15: Security Profiles

Part 16: DICOM Content Mapping Resource

Part 17: Explanatory Information

Part 18: Web Services

Part 19: Application Hosting

Part 20: Transformation of DICOM to and from HL7 Standards

Part 21: Transformations between DICOM and other Representations

Part 22: Real-Time Communication

A Quick Walk Through DICOM

Part 1 of the standard gives an overview of the standard. Since this part was approved before most of the other parts were completed, it is already somewhat outdated and can be confusing.

Part 2 describes DICOM conformance and how to write a conformance statement. A conformance statement is important because it allows a network administrator to plan or coordinate a network of DICOM applications. For an application to claim DICOM conformance, it must have an accurate conformance statement.

Parts 3 and 4 define the types of services and information that can be exchanged using DICOM.

Parts 5 and 6 describe how commands and data shall be encoded so that decoding devices can interpret them.

Part 7 describes the structure of the DICOM commands that, along with related data, make up a DICOM message. This part also describes the association negotiation process, where two DICOM applications mutually agree on the services they will perform over the network.

Part 8 describes how the DICOM messages are exchanged over the network using two prominent transport layer protocols: TCP/IP and OSI. (Note that IPv4 and IPv6 are supported by DICOM and by Merge DICOM Toolkit.) This is termed the DICOM Upper Layer Protocol (DICOM UL).

Part 9 describes how DICOM messages shall be exchanged using the 'old' 50-pin point-to-point connection originally specified in the predecessor to DICOM (ACR/NEMA Version 2). This part has been retired from the DICOM standard.

Part 10 describes the DICOM model for the storage of medical imaging information on removable media. It specifies the contents of a DICOM File Set, the format of a DICOM File and the policies associated with the maintenance of a DICOM Media Storage Directory (DICOMDIR) structure.

Part 11 specifies the Media Storage Application Profiles that standardize a number of choices related to a specific clinical need (modality or application). This includes the specification of a specific physical medium and media format (e.g., CD-ROM, 3.5" high-density floppy, ...), as well as the types of information (objects) that can be stored within the DICOM File Set. Part 11 also includes useful templates to provide guidance in authoring media application conformance statements.

Part 12 details the characteristics of various physical medium and media formats that are referenced by the Media Storage Application Profiles of Part 11.

While parts 11 and 12 of DICOM are expected to evolve along with the introduction of new clinical procedures and the advancement of storage media and file system technology, Part 10 should remain quite stable since it specifies file formats independent of medical application or storage technology.

Part 13 details a point-to-point protocol for doing print management services. This part has been retired from the DICOM standard.

Part 14 specifies a standardized display function for displaying grayscale images.

Part 15 specifies Security Profiles to which implementations may claim conformance. Profiles are defined for secure network transfers and secure media.

Part 16 specifies the DICOM Content Mapping Resource (DCMR) which defines the templates and context groups used elsewhere in the standard.

Part 17 consolidates informative information previously contained in other parts of the standard. It is composed of several annexes describing the use of the standard.

Part 18 specifies a web-based service for accessing and presenting DICOM persistent objects (e.g. images, medical imaging reports).

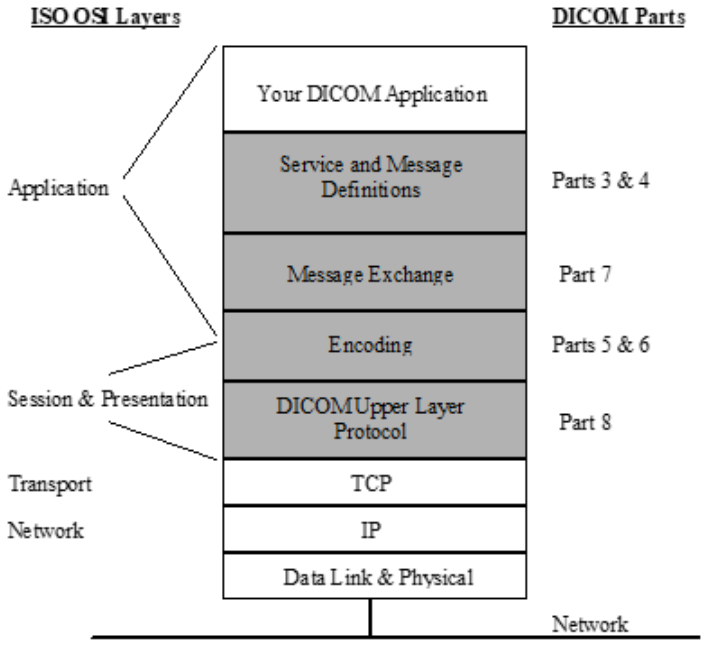
Part 19 defines an API such that a 'plug-in' Hosted Application written to the API would be able run in any environment provided by a Hosting System implementing the API.

Part 20 specifies transformations of DICOM data to and from HL7 standards.

Part 21 specifies the transformations between DICOM and other representations of the same information.

Part 22 specifies an SMPTE ST 2110-10 based service, relying on RTP, for the real-time transport of DICOM metadata. It provides a mechanism for the transport of DICOM metadata associated with a video or an audio flow based on the SMPTE ST 2110-20 and SMPTE ST 2110-30, respectively.

The figure below maps portions of the DICOM Standard dealing with networking to the ISO Open Systems Interconnection (OSI) basic reference model. The organization and terminology of the DICOM Standard corresponds closely with that used in the OSI Standard.



Where to get the DICOM Standard

As a user of this toolkit, you should have access to the DICOM Standard. The Merge DICOM Toolkit takes care of most of the details of DICOM for you. However, the standard is the final word. You will probably find Parts 2 - 6 most useful. The DICOM Standard can be ordered from:

NEMA

1300 N. 17th Street

Suite 1847

Rosslyn, VA 22209

USA
http://dicomstandard.org

The DICOM Standard is typically published every other year. Each version includes approved changes since the last publishing. The most recent version of the standard is available in PDF format and can be downloaded from NEMA's public ftp site at: ftp://medical.nema.org/medical/Dicom.

Special Note

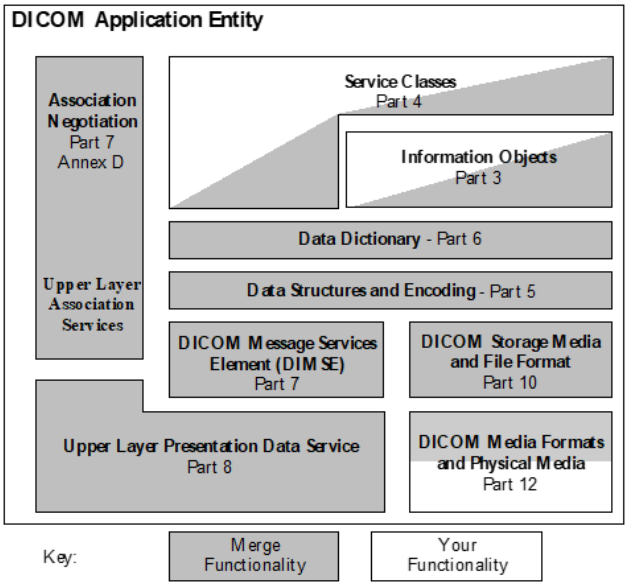
Note that the DICOM Standard is evolving so rapidly that additions to the Standard are published as 'supplements'. For example, the media extensions have been incorporated into the DICOM Standard as a supplement that contains addenda to various parts of the standard (e.g., PS3.3, PS3.4, ...). If you find that this document references a part of the Standard which you cannot find, obtain the proper supplement from NEMA. Other additions to the Standard (e.g., new image objects or documents) are also published as supplements. NEMA also makes all supplements to the standard freely available on their FTP server. You can reference these supplements at:

ftp://medical.nema.org/medical/Dicom/Final/.

1.2. The Merge DICOM Toolkit

The Merge DICOM Toolkit provides a generalized implementation of DICOM in a Python Library which you use with your application. You make simple function calls to open connections with other DICOM devices on a network, and to build and exchange DICOM messages or DICOM files.

The figure below presents a pictorial representation of a DICOM Application Entity. The Merge DICOM Toolkit implements Parts 5, 6, 7, 8, and 10 of the DICOM Standard. It also makes it much easier for your application to implement according to Parts 3 and 4 by supplying many tools for the management of DICOM messages, and to Part 12 by supplying 'hooks' to your application's underlying file system.



The DICOM Toolkit also supplies useful utility programs for testing a DICOM network connection, creating sample DICOM messages and writing them to a file, and validating and listing the contents of DICOM messages.

The DICOM Standard and the Merge DICOM Toolkit allow applications to add private information to a DICOM message or file. For most application developers, this is more than sufficient. For applications that need to define their own non-standard private network or file services, an optional Merge DICOM Toolkit Extended Toolkit is available.

1.3. Development Platform Requirements

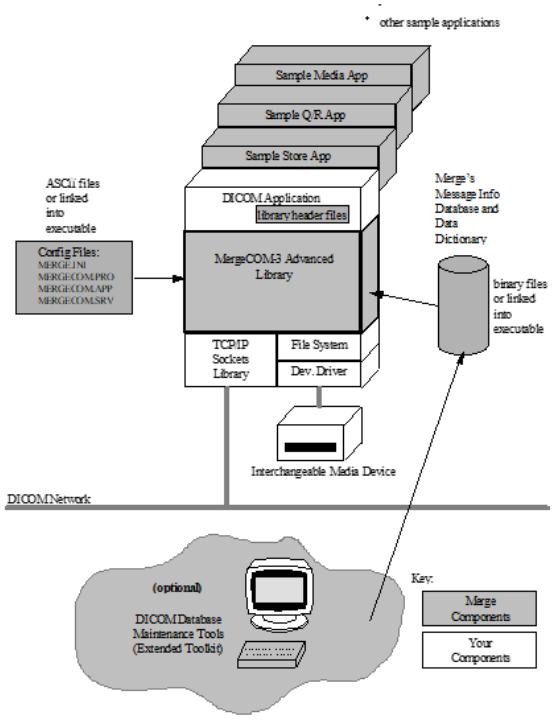
To use the Merge DICOM Toolkit Library, you must run on a toolkit supported computing platform. The DICOM Python Toolkit is available for Linux x64 (Python version 3.6) and Windows 10 x64 (Python version 3.8) platforms. If it is not currently available for your target platform, please contact Merge Healthcare. We may already be working on the port.

Your development environment (or at a minimum your target environment) should run on a machine with a network interface over which you can run the TCP/IP protocol. The DICOM Toolkit library supplies you with the DICOM protocol that runs on top of TCP/IP.

If your application will write DICOM files to interchangeable media, you will need a device driver for the media storage device and a programming interface between your operating system and the file system on that device.

1.4. Library Structure

The organization and components of the Merge DICOM Toolkit Library are shown in the figure below.



1.4.1. Merge DICOM Toolkit Library

The Merge DICOM Toolkit library has been carefully designed to be re-entrant and has been validated to be thread-safe on several multi-threading capable platforms. Note, however, that with only a few exceptions, Merge DICOM Toolkit assumes that objects are only accessed from one thread at a time. For instance, Merge DICOM Toolkit assumes that only a single thread will manipulate a message object at one time.

Shared libraries or dynamic link libraries (DLL's) are normally supplied by Merge DICOM Toolkit for platforms which support them.

When a Merge DICOM Toolkit Application is first run, it reads in its configuration files; usually named `merge.ini`, `mergecom.app`, `mergecom.pro`, and `mergecom.srv`. Toolkit configuration is described later in this document. Usually, it is desirable to keep these configurable parameters in ASCII files for easy modification. When modifying your configuration files, your application must be re-run for those changes to take effect.

In cases where the toolkit configuration is unlikely to be changed or it is desirable to make these changes within the running application, the toolkit configuration can be compiled into your application. Most configurable parameters can be dynamically modified and reset within your running application.

The Merge DICOM Python Toolkit with Python code includes C/C++ extension libraries `mc3adv.pyd` (Window 10 x64) and `mc3adv.so` (Linux x64). In the process of servicing networking calls, these libraries require the services of a Berkeley Sockets (or WinSock) Library for your platform.

1.4.2. Binary Message Information and Data Dictionary Files

A great deal of the power of Merge DICOM Toolkit lies in its message handling and message validation capabilities. Message Objects are what is communicated between DICOM Application Entities. When your application creates a DICOM message object, the library accesses a binary message info file with information about that class of message. This info file describes to the library what attributes to expect as part of that message and each attribute's characteristics (Value Type, Conditions, and Enumerated or Defined Terms).

Another binary file containing the data dictionary is also accessed by the library. The data dictionary contains other characteristics of attributes (Name, Value Representation, and Value Multiplicity).

Performance Tuning

Merge DICOM Toolkit gives you added flexibility, by not requiring your application to make use of the message info file. Certain API calls allow you to open messages without accessing the info files. This means that the toolkit cannot validate your message against the DICOM standard, but this may not always be necessary once an application becomes stable. These options are discussed in greater detail in the Developing DICOM Applications section of this document.

Two specialized classes (subclasses) of message objects are also supported by the DICOM Toolkit Library: items and files. Items are DICOM 'sub-messages' that can be stored in a DICOM message within a sequence of items. DICOM files are specialized DICOM messages that contain additional file meta-information and are written to or read from interchangeable media rather than transmitted or received over a network. Most Merge DICOM Toolkit API calls dealing with message objects can also operate on items and files (these calls would be called polymorphic in object-oriented parlance). DICOM messages, items, and files will be described in much greater detail later in this document.

1.4.3. Sample Applications

Included with the toolkit are sample applications in Python source code. Samples include SCP/SCU client and server applications supplied for Storage. Also, a compression sample and DICOM File Service application are provided.

Before writing your own applications, check the sample source. While these sample applications are primitive in features and user interface, they illustrate how to use the DICOM Toolkit API to perform DICOM services over a network.

1.4.4. Merge DICOM Toolkit Extended Toolkit

Merge Healthcare has a DICOM Database Management System in which the DICOM standard is maintained. This database, along with a few additional tools, is used to generate the binary message info and dictionary files accessed by the DICOM Toolkit. As the DICOM standard is updated or extended, by simply maintaining this database, we can generate new binary files and keep the toolkit current. This also reduces the number of changes that must be made in the core DICOM Toolkit library over time.

The extended version of this toolkit makes some of these tools available to application developers who need to significantly extend the standard with private attribute and private service definitions. The files for the extended version are packaged with the standard toolkit. The extended version supplies you with an ASCII file database of the standard that you can extend, along with executables for your platform that translate these ASCII files to the binary message info and data dictionary files used by the toolkit at run time. In this way, you can extend the toolkit to validate your own private attributes and services.

1.5. Conventions

This manual follows a few formatting conventions.

Terms that are being defined are presented in **boldface**.

Sample commands appear in **`bold courier`** font, while sample output , source code, and function calls appear in `standard courier` font.

Hexadecimal numbers are written with a trailing H. For example 16 decimal is equivalent to 10H hexadecimal.

Chapter 2. Understanding DICOM

The eighteen separate parts of the DICOM Standard can seem overwhelming, and most would agree that they are difficult to read. Part of what makes a successful standard is precision and detail. Our goal here is to explain the key concepts without delving too far into the detail, most of which is handled automatically for you by the DICOM Toolkit.

2.1. General Concepts

Some key concepts that must be understood to use the DICOM Toolkit wisely are common across both DICOM networking and interchangeable media applications. These concepts are discussed first.

2.1.1. Application Entities

The DICOM Standard refers extensively to **Application Entities (AEs)**. An application entity is simply a DICOM application. If your application interacts with other applications on a network or with interchangeable media using the DICOM protocol, it is an application entity.

DICOM also refers to **Service Class Users (SCUs)** and **Service Class Providers (SCPs)**. An application entity is an SCU when it requests DICOM services over a network and an SCP when it provides DICOM services over a network. We will more often refer to the SCU as a **Client** and the SCP as a **Server**. A single DICOM application entity can act as both a client and a server. This client/server model is a powerful and omnipresent one in the world of distributed network computing.

2.1.2. Services and Meta Services

DICOM is formed around the concepts of **Services** and **Service Classes**. The DICOM Standard specifies a set of services that can be performed over a network. Some of the services can also be stored to interchangeable media (these are italicized in the table below). As new services are introduced, the standard will be further expanded. The standard also groups related services into a service class. The table below lists the DICOM standard service classes and their component services.

When a particular collection of services in a service class implies a higher level of service, this collection is combined by the standard into a **Meta Service**. Specifying that your application supports a specific meta service is a useful shorthand for explicitly listing out the collection of services that make up that meta service. [2]

[2] The DICOM Standard actually refers to services as Service Object Pairs (SOPs) and meta services as Meta-SOPs. We avoid this terminology to avoid unnecessary detail and confusion.

Table 2.1: DICOM Services Classes and their Component Services

Service Class	Services	Description
Verification	Verification	Verifies application level communication between DICOM application entities (AE's).
Storage	12-lead ECG Waveform Acquisition Context SR Advanced Blending Presentation State Ambulatory ECG Waveform Arterial Pulse Waveform Audio Waveform Real-Time Communication Autorefraction Measurements Basic Structured Display Basic Text SR Basic Voice Audio Waveform Blending Softcopy Presentation State Body Position Waveform Breast Projection X-Ray Image - For Presentation Breast Projection X-Ray Image - For Processing Breast Tomosynthesis Image C-Arm Photon Electron Radiation Record C-Arm Photon-Electron Radiation Cardiac Electrophysiology Waveform Chest CAD SR Colon CAD SR Color Palette Color Softcopy Presentation State Compositing Planar MPR Volumetric Presentation State Comprehensive SR Comprehensive 3D SR Computed Radiography Image Content Assessment Results Corneal Topography Map CT Defined Procedure Protocol CT Image CT Performed Procedure Protocol Deformable Spatial Registration Dermoscopic Photography Image Digital Intra-oral X-Ray Image - For Presentation Digital Intra-oral X-Ray Image - For Processing Digital Mammography Image - For Presentation Digital Mammography Image - For Processing Digital X-Ray Image - For Presentation Digital X-Ray Image - For Processing	Transfer of medical images and related standalone data between DICOM application entities, either over a network or using interchangeable media.

Service Class	Services	Description
	Electromyogram Waveform Electrooculogram Waveform Encapsulated CDA Encapsulated MTL Encapsulated OBJ Encapsulated PDF Encapsulated STL Enhanced Continuous RT Image Storage Enhanced CT Image Enhanced MR Color Image Enhanced MR Image Enhanced PET Image Enhanced RT Image Storage Enhanced SR Enhanced US Volume Enhanced X-Ray Radiation Dose SR Enhanced XA Image Enhanced XRF Image Extensible SR General Audio Waveform General ECG Waveform Generic Implant Template Grayscale Planar MPR Volumetric Presentation State Grayscale Softcopy Presentation State Hanging Protocol Hardcopy Color Image Hardcopy Grayscale Image Hemodynamic Waveform Implant Assembly Template Implant Template Group Implantation Plan SR Document Intraocular Lens Calculations Intravascular Optical Coherence Tomography Image - For Presentation Intravascular Optical Coherence Tomography Image - For Processing Inventory Keratometry Measurements Key Object Selection Document Legacy Converted Enhanced CT Image Legacy Converted Enhanced MR Image Legacy Converted Enhanced PET Image Lensometry Measurements Macular Grid Thickness and Volume Report Mammography CAD SR	

Service Class	Services	Description
	Microscopy Bulk Simple Annotations MR Image MR Spectroscopy Multi-channel Respiratory Waveform Multi-frame Grayscale Byte Secondary Capture Image Multi-frame Grayscale Word Secondary Capture Image Multi-frame Single Bit Secondary Capture Image Multi-frame True Color Secondary Capture Image Multiple Volume Rendering Volumetric Presentation State Nuclear Medicine Image Ophthalmic 16 bit Photography Image Ophthalmic 8 bit Photography Image Ophthalmic Axial Measurements Ophthalmic Optical Coherence Tomography B-scan Volume Analysis Ophthalmic Optical Coherence Tomography En Face Image Ophthalmic Thickness Map Ophthalmic Tomography Image Ophthalmic Visual Field Static Perimetry Measurements Parametric Map Patient Radiation Dose SR Performed Imaging Agent Administration SR Planned Imaging Agent Administration SR Positron Emission Tomography Image Procedure Log Protocol Approval Pseudo-Color Softcopy Presentation State Radiopharmaceutical Radiation Dose SR Raw Data Real World Value Mapping Rendition Selection Document Real-Time Communication Respiratory Waveform Robotic-Arm Radiation Robotic-Arm Radiation Record Routine Scalp Electroencephalogram Waveform RT Beams Delivery Instruction RT Beams Treatment Record RT Brachy Application Setup Delivery Instruction RT Brachy Treatment Record RT Dose RT Image RT Ion Beams Treatment Record RT Ion Plan	

Service Class	Services	Description
	RT Patient Position Acquisition Instruction Storage RT Physician Intent RT Plan RT Radiation Record Set RT Radiation Salvage Record RT Radiation Set Delivery Instruction RT Radiation Set RT Segment Annotation RT Structure Set RT Treatment Preparation RT Treatment Summary Record Secondary Capture Image Segmentation Segmented Volume Rendering Volumetric Presentation State Simplified Adult Echo SR Sleep Electroencephalogram Waveform Spatial Fiducials Spatial Registration Spectacle Prescription Report Standalone Curve Standalone Modality LUT Standalone Overlay Standalone PET Curve Standalone VOI LUT Stereometric Relationship Stored Print Subjective Refraction Measurements Surface Scan Mesh Surface Scan Point Cloud Surface Segmentation Tomotherapeutic Radiation Record Tomotherapeutic Radiation Tractography Results Ultrasound Image Ultrasound Multi-Frame Image Variable Modality LUT Softcopy Presentation State Storage Video Endoscopic Image Video Endoscopic Image Real-Time Communication Video Microscopic Image Video Photographic Image Video Photographic Image Real-Time Communication Visual Acuity Measurements	

Service Class	Services	Description
	VL Endoscopic Image VL Microscopic Image VL Photographic Image VL Slide-Coordinates Microscopic Image VL Whole Slide Microscopy Image Volume Rendering Volumetric Presentation State Wide Field Ophthalmic Photography 3D Coordinates Image Wide Field Ophthalmic Photography Stereographic Projection Image X-Ray 3D Angiographic Image X-Ray 3D Craniofacial Image X-Ray Angiographic Image X-Ray Angiographic Bi-Plane Image X-Ray Radiation Dose SR X-Ray Radiofluoroscopic Image XA/XRF Grayscale Softcopy Presentation State XA Defined Procedure Protocol XA Performed Procedure Protocol	
Storage Commitment	Storage Commitment Push Storage Commitment Pull	Ensures that SOP Instances stored with the storage service class will not be deleted after reception but will be stored safely and can be retrieved again at a later point.
Storage Management	Inventory Creation	An application-level class-of-service that facilitates peer-to-peer controls for management of persistent storage of Composite SOP Instances.
Media Storage	DICOM Basic Directory Storage and storage of various (italicized) services from the other Service Classes	Exists as a member of every DICOM File Set and contains general information about the file set and a hierarchical directory of the DICOM files contained in the file set.

Service Class	Services	Description
Query/ Retrieve	Defined Procedure Protocol Information Model Find Defined Procedure Protocol Information Model Move Defined Procedure Protocol Information Model Get Inventory Find Inventory Get Inventory Move Patient Root Find Patient Root Move Patient Root Get Patient/Study Only Find (Retired) Patient/Study Only Move (Retired) Patient/Study Only Get (Retired) Protocol Approval Information Model Find Protocol Approval Information Model Move Protocol Approval Information Model Get Repository Query Study Root Find Study Root Move Study Root Get	Management of images through a query and retrieval mechanism based on a small number of key attributes.
Basic Worklist Management	Modality Worklist Find	Supports the exchange of any type of worklist from one AE to another.
Print Management	Basic Annotation Box Basic Color Image Box Basic Film Session Basic Film Box Basic Grayscale Image Box Basic Print Image Overlay Box Image Overlay Box Retired Presentation LUT Print Job	Printing (or filming) of medical images and image related data on a hard copy medium. Also, storage of print related data to interchangeable media.
	Print Queue Management Printer Printer Configuration Retrieval Printer Referenced Image Box Pull Print Request VOI LUT Box **Basic Grayscale Print Mgmt. Meta** **Basic Color Print Mgmt. Meta** **Pull Stored Print Mgmt. Meta** **Ref. Grayscale Print Mgmt. Meta** **Ref. Color Print Mgmt. Meta**	

Service Class	Services	Description
Study Content Notification	Basic Study Content Notification	Allows one DICOM AE to notify another DICOM AE of the existence, contents, and source location of the images of a study.
Patient Management	Detached Patient Management Detached Visit Management **Detached Patient Mgmt. Meta**	Creation and tracking of the subset of patient and patient visit information that is required to aid in the management of radiographic studies.
Study Management	Detached Study Management Study Component Management Modality Performed Procedure Step Modality Performed Procedure Step Notification Modality Performed Procedure Step Retrieve	Creation, scheduling, performance, and tracking of imaging studies.
Results Management	Detached Results Management Detached Interpretation Management **Detached Results Mgmt. Meta**	Creation and tracking of results and associated diagnostic interpretations.

2.1.3. DICOM Information Model

The DICOM Standard includes the specification of a **DICOM Information Model**. A detailed entity-relationship diagram of this model is included in both parts 3 and 4 of the standard. This model specifies the relationship between the different types of objects (also called entities) managed in DICOM. For example, a Patient has one or more Studies, each of which are composed of one or more Series and zero or more Results, etc.

a. Objects vs. Object Instances

Most of DICOM's services perform actions on or with **object instances**.[3] An **object** can be thought of as a class of data (CT Image, Film Box, etc.) while an object instance is an actual occurrence of an object (a particular CT Image, a populated Film Box, etc.).

[3] object instances are referred to as SOP Instances or managed SOPs in the DICOM standard.

b. Normalized vs. Composite

There are two types of objects (and hence, object instances) defined in DICOM. **Normalized objects** are objects consisting of a single entity in the DICOM information model (e.g., a Film Box). **Composite objects** are composed of several related entities (e.g., an MR Image). When possible, it is preferable to deal with normalized object instances over the network, because they contain less redundant data and can be more efficiently managed by an application.

Most services inherited from the ACR/NEMA Version 2.x Standard are **composite services** (operate on composite object instances) for reasons of backward compatibility. Newly introduced services, such as the HIS/RIS and Print Management Services, tend to be **normalized services** (operate on normalized object instances).

2.2. Networking

Certain aspects of DICOM only apply to networking when using the DICOM Toolkit. This includes networking commands and association negotiation.

2.2.1. Commands

DICOM defines a set of networking **commands**[4]. Each service uses a subset of these DICOM commands to perform the service over a network. These commands usually act on object instances. The C-commands operate on composite object instances, while the N-commands operate on normalized object instances.

[4] commands are referred to as DIMSE Services in the DICOM Standard.

The DICOM commands and brief descriptions of their actions are listed in the table below.

Table 2.2: DICOM Commands

DICOM Commands	Description
C-STORE	Transfer an object instance to a remote AE.
C-GET	Retrieve from a remote AE object instance(s) whose attributes match a specified set of attributes.
C-MOVE	Move object instance(s) whose attributes match a specified set of attributes from a remote AE to yet another remote AE (or possibly your own AE - which would be another form of retrieval).
C-FIND	Match a set of attributes to the attributes of a set of object instances on a remote AE.
C-ECHO	Verify end-to-end communications with a remote AE.
N-EVENT-REPORT	Report an event to a remote AE.
N-GET	Retrieve attribute values from a remote AE.
N-SET	Request modification of attribute on a remote AE.
N-ACTION	Request an action by a remote AE.
N-CREATE	Request that a remote AE create a new object instance.
N-DELETE	Request that a remote AE delete an existing object instance.

These DICOM commands can be thought of as primitives that every networking service is built from. In the context of a particular Service, these primitive actions translate to explicit real-world activities on the part of an Application Entity. Hence, DICOM places requirements on an application implementing a DICOM service. DICOM is careful to only express high-level operational requirements, and leaves the creative detail and look and feel of the application entity to the developer.

a. Request vs. Response

For every command, there is both a **request** and a **response**. A command request indicates that a command should be performed and is usually sent to an SCP. A command response indicates whether a command completed or its state of completion and is usually returned to an SCU. Example request commands are C STORE-RQ, N-GET-RQ, and N-SET-RQ. Example response commands are C STORE-RSP, N-GET-RSP, and N-SET-RSP.

NOTE: It is important to note that this service definition level is where the Merge DICOM Toolkit Library leaves off, and your Application begins. While Merge DICOM Toolkit supplies sample application guides and running sample application source code for your platform, they are only supplied as an example. They clearly explain the requirements that implementing certain DICOM services places on your application and provide worthwhile but primitive examples of how to approach your application with the toolkit. While you will see that the toolkit saves you a great deal of 'DICOM work', it does not implement your end application for you.

2.2.2. Association Negotiation

One of the two areas where Merge DICOM Toolkit does a great deal of the 'DICOM work' for you is in opening an association (session) with another DICOM AE over the network. DICOM application entities need to agree on certain things before they operate with one another (open an association); these include:

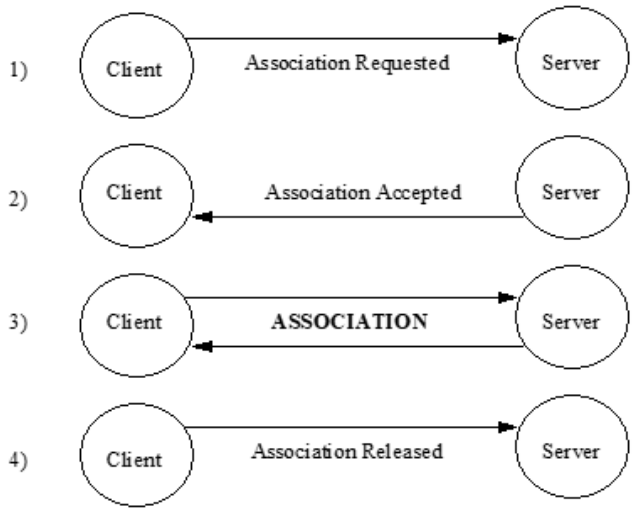
- the services that can be performed between the two devices, which also impacts the commands and object instances that can be exchanged.
- the **transfer syntax** that shall be used in the network communication. The transfer syntax defines how the commands and object instances are encoded 'on the wire'.

The exchange of DICOM commands and object instances can only occur over an open association.

DICOM defines an association negotiation protocol (see the figure below). In the most common DICOM services, a client application entity (SCU) proposes an association with a server AE (SCP). However, some services define a mechanism where the client can be the SCP which opens an association with the SCU. This is used when an SCP sends asynchronous event reports to an SCU through the N EVENT REPORT command. This is done through DICOM role negotiation, which is used during standard association negotiation. For the sake of simplicity, the remainder of this manual refers to the client as the SCU and the server as the SCP.

The association request proposal contains the set of services the client would like to perform and the transfer syntaxes it understands. The server then responds to the client with a subset of the services and transfer syntaxes proposed by the client. If this subset is empty, the server has rejected the association. If the subset is not empty, the server has accepted the association and the agreed upon services may be performed.

The client is responsible for releasing the association when it is finished performing its network operations. Either the client or the server can abort the association in the case of some catastrophic failure (e.g., disk full, out of memory).



2.3. Messages

Service-Command Pair

Once an association is established, services are performed by AEs through the exchange of DICOM **Messages**. A message is the combination of a DICOM command request or response and its associated object instance (see the figure below). Messages containing command requests will be referred to as request messages, while messages containing command responses will be referred to as **response messages**.

When a DICOM service is stored to interchangeable media in a DICOM File, the structure of a DICOM File is a slightly specialized class of DICOM message. Media interchange is discussed in detail later; the only important thing to realize for now is that much of what is discussed relating to DICOM Messages also applies to DICOM Files.

DICOM specifies the required message structure for each **service-command pair**. For example, the Patient Root Find - C-FIND-RQ service-command pair has a specific message structure. The command portion of a message is specified in Part 7 of the standard, while the object instance portion is specified in Parts 3 and 4.

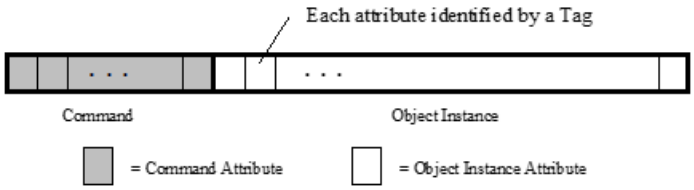
Attributes, Values and Tags

A message is constructed of **attributes** having **values**, with each **attribute** identified by a **tag**. An attribute is a unit of data (e.g., Patient's Name, Scheduled Discharge Date, ...). A tag is a 4 byte number identifying an attribute (e.g., 00100010H for Patient's Name, 0038001CH for Scheduled Discharge Date, ...).

Groups and Elements

A tag is usually written as an ordered pair of two byte numbers. The first two bytes are sometimes called a **group** number, with the last two bytes being called an **element** number (e.g., (0010, 0010), (0038, 001C), ...). This terminology is partly a remnant of the ACR-NEMA Standard where elements within a group were related in some manner. This can no longer be depended on in DICOM, but the ordered pair notation is still useful and often easier to read.

Also, the ordered pair notation is important when defining a Tag for a private attribute. We will see later that all private attributes must have an odd group number.



2.3.1. DICOM Data Dictionary

Attributes have certain characteristics that apply to them no matter what message they are used in. These characteristics are specified in the DICOM Data Dictionary (Part 6 of DICOM) and are **Value Representation (VR)** and **Value Multiplicity (VM)**.

Value Representation can be thought of as the 'type specifier' for the values that can be assigned to an attribute. This includes the data type, as well as its format. The VRs defined by DICOM are listed in the table below. You should refer to Part 5 of the standard for a detailed description of their allowed values and formats.

Table 2.3: DICOM Value Representations (VR's)

VR	Name	VR	Name
AE	Application Entity	OW	Other Word
AS	Age String	PN	Person Name
AT	Attribute Tag	SH	Short String
CS	Code String	SL	Signed Long
DA	Date	SQ	Sequence of Items
DS	Decimal String	SS	Signed Short
DT	Date Time	ST	Short Text
FL	Floating Point Single	SV	Signed 64-bit Very Long
FD	Floating Point Double	TM	Time
IS	Integer String	UC	Unlimited Characters
LO	Long String	UI	Unique Identifier
LT	Long Text	UL	Unsigned Long
OB	Other Byte	UN	Unknown
OD	Other Double	UR	URI or URL
OF	Other Float	US	Unsigned Short

VR	Name	VR	Name
OL	Other Long	UT	Unlimited Text
OV	Other 64-bit Very Long	UV	Unsigned 64-bit Very Long

A single attribute can have multiple values. Value Multiplicity defines the number of values an attribute can have. VM can be specified as 1, k , 1-k or 1-n, where k is some integer value and n represents 'many'. For example, Part 6 specifies the VM of Scheduled Discharge Time (0038, 001D) as 1, while the VM of Referenced Overlay Plane Groups (2040, 0011) is 1-99.

2.3.2. Message Handling

Given the number of services and commands specified in TABLE 2.1: DICOM SERVICES CLASSES AND THEIR COMPONENT SERVICES ON PAGE 15 and TABLE 2.2: DICOM COMMANDS ON PAGE 22, it is clear that there are a great deal of messages to manage in DICOM. Remember, each service-command pair implies a different message. Fortunately, you will see later that Merge DICOM Toolkit saves the application developer a great deal of work in the message handling arena.

DICOM specifies the required contents of each message in Parts 3, 4, and 7 of the standard. For each attribute included in a message, additional characteristics of the attribute are defined that only apply within the context of a service. These characteristics are **Enumerated Values**, **Defined Terms**, and **Value Type**.

DICOM specifies that some attributes should have values from a specified set of values. If the attribute is an enumerated value, it shall have a value taken from the specified set of values. A good example of enumerated values are (M, F, O) for Patient's Sex (0010, 0040) in Storage services. If the attribute is a defined term, it may take its value from the specified set, or the set may be extended with additional values. An example of defined terms are (CREATED, RECORDED, TRANSCRIBED, APPROVED) for Interpretation Status ID (4008, 0212) in Results Management services. If this set is extended by an application with another term, such as IN PROCESS, it should be documented in that application's conformance statement.

The most important characteristic of an attribute that is specified on a message by message basis, is the Value Type (VT). The VT of an attribute specifies whether or not that attribute needs to be included in a message and if it needs to have a value. Attributes can be required, optional, or only required under certain conditions (conditional attributes). Conditional attributes are always specified along with a condition. The value types defined by DICOM are listed in the table below. Note that a null valued attribute has a value, that value being null (zero length).

Table 2.4: DICOM Value Types (VT's)

Value Type (VT)	Description
1	The attribute must have a value and be included in the message. The value cannot be null (empty).
1C	The attribute must have a value and be included in the message only under a specified condition. The value cannot be null. If that condition is not met, the attribute shall not be included in the message.
2	The attribute must have a value and be included in the message. If the value for the attribute is unknown and cannot be specified, its value shall be null.

Value Type (VT)	Description
2C	The attribute must have a value and be included in the message only under a specified condition. If the value for the attribute is unknown and cannot be specified, its value shall be null. If that condition is not met, the attribute shall not be included in the message
3	The attribute is optional. It may or may not be included in the message. If included, the attribute may or may not have a null value.

2.3.3. Private Attributes

The DICOM Standard allows application developers to add their own private attributes to a message as long as they are careful to follow certain rules. A **private attribute** is identified differently than are standard attributes. Its tag is composed of an odd group number, a private identification code string, and a single byte element number.

For example, ACME Imaging Inc. might define a private attribute to hold the name of the field engineer that last serviced their equipment. They could assign this attribute to private attribute tag (1455, 'ACME_IMG_INC', 00). This attribute has group number 1455, a private identification code string of 'ACME_IMG_INC', and a single byte element number of 00.

ACME could assign up 255 other private attributes to private group 1455 by using the other element numbers (01-FF). Part 5 of DICOM explains how these private tags are translated to standard group and element numbers and encoded into a message, while avoiding collisions. Merge DICOM Toolkit handles these details for you.

DICOM makes a couple of rules that must be followed when using private attributes:

- Private attributes shall not be used in place of required (Value Type 1, 1C, 2, or 2C) attributes.
- The possible value representations (VRs) used for private attributes shall be only those specified by the standard (see TABLE 2.4: DICOM VALUE TYPES (VT'S) ON PAGE 26).

The way you use private attributes in your application can also greatly affect your conformance statement. DICOM conformance is discussed in greater detail later.

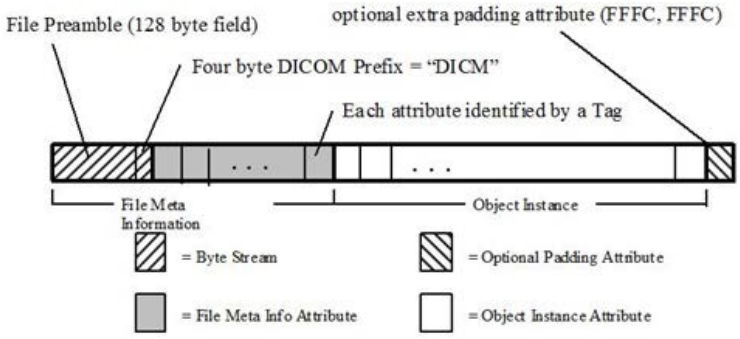
2.4. Media Interchange

The DICOM Standard specifies a DICOM file format for the interchange of medical information on removable media. This file format is a logical extension of the networking portion of the standard. When an object instance that was communicated over a network would also be of value when communicated via removable media, DICOM specifies the encapsulation of these object instances in a DICOM file.

2.4.1. DICOM Files

A **DICOM File** is the encapsulation of a DICOM object instance, along with **File Meta Information**. File meta information is stored in the header of every DICOM file and includes important identifying

information about the encapsulated object instance and its encoding within the file (see the figure below).



The file meta information begins with a 128 byte buffer available for application profile or implementation specific use. **Application Profiles** standardize a number of choices related to a specific clinical need (modality or application) and are specified in Part 11 of the DICOM Standard. The next four bytes of the meta information contain the DICOM prefix, which is always "DICM" in a DICOM file and can be used as an identifying characteristic for all DICOM files. The remainder of the file (preamble and object instance) is encoded using tagged attributes (as in a DICOM Message).

The object instances that can be stored within the DICOM file are equivalent to a subset of the object instances that can be transmitted in network messages. The services that can be performed to interchangeable media are *italicized* in TABLE 2.1: DICOM SERVICES CLASSES AND THEIR COMPONENT SERVICES ON PAGE 15. The Media Storage Service Class (in Part 4 of the DICOM standard) specifies which service-command pairs can be performed to media. Remember it is the service command pair that identifies the object instance portion of the message, and it is only the object instance portion of the message that is stored in a DICOM file. The command attributes associated with a network message are never stored in a DICOM File.

The service command pairs whose corresponding object instances can be stored to media are summarized in the table below.

NOTE: The Media Storage Directory Service is not performed over a network and the single object specified in the Basic Directory Information Object Definition (Part 3) is used.

Table 2.5: Service-Command Pairs Specifying Object Instances that can be Stored in a DICOM File

Service	Command
12-lead ECG Waveform Storage	C-STORE
Advanced Blending Presentation State Storage	C-STORE
Ambulatory ECG Waveform Storage	C-STORE
Arterial Pulse Waveform Storage	C-STORE
Audio Waveform Real-Time Communication	C-STORE
Autorefraction Measurements Storage	C-STORE
Basic Color Image Box	N-SET
Basic Film Box	N-CREATE

Service	Command
Basic Film Session	N-CREATE
Basic Grayscale Image Box	N-SET
Basic Structured Display Storage	C-STORE
Basic Text Structured Reporting	C-STORE
Basic Voice Audio Waveform Storage	C-STORE
Blending Softcopy Presentation State Storage	C-STORE
Body Position Waveform Storage	C-STORE
Breast Projection X-Ray Image Storage - For Presentation	C-STORE
Breast Projection X-Ray Image Storage - For Processing	C-STORE
Breast Tomosynthesis Image Storage	C-STORE
C-Arm Photon-Electron Radiation Record Storage	C-STORE
C-Arm Photon-Electron Radiation Storage	C-STORE
Cardiac Electrophysiology Waveform Storage	C-STORE
Chest CAD SR	C-STORE
Colon CAD SR	C-STORE
Color Palette Storage	C-STORE
Color Softcopy Presentation State Storage	C-STORE
Comprehensive Structured Reporting	C-STORE
Computed Radiography Image Storage	C-STORE
CT Image Storage	C-STORE
Deformable Spatial Registration Storage	C-STORE
Dermoscopic Photography Image Storage	C-STORE
Detached Interpretation Management	N-GET
Detached Patient Management	N-GET
Detached Results Management	N-GET
Detached Study Management	N-GET
Detached Study Component Management	N-GET
Detached Visit Management	N-GET
Digital Intra-oral X-Ray Image Storage - For Presentation	C-STORE
Digital Intra-oral X-Ray Image Storage - For Processing	C-STORE

Service	Command
Digital Mammography Image Storage - For Presentation	C-STORE
Digital Mammography Image Storage - For Processing	C-STORE
Digital X-Ray Image Storage - For Presentation	C-STORE
Digital X-Ray Image Storage - For Processing	C-STORE
Electromyogram Waveform Storage	C-STORE
Electrooculogram Waveform Storage	C-STORE
Encapsulated CDA Storage	C-STORE
Encapsulated MTL Storage	C-STORE
Encapsulated OBJ Storage	C-STORE
Encapsulated PDF Storage	C-STORE
Encapsulated STL Storage	C-STORE
Enhanced Continuous RT Image Storage	C-STORE
Enhanced CT Image Storage	C-STORE
Enhanced MR Color Image Storage	C-STORE
Enhanced MR Image Storage	C-STORE
Enhanced PET Image Storage	C-STORE
Enhanced RT Image Storage	C-STORE
Enhanced Structured Reporting	C-STORE
Enhanced US Volume Storage	C-STORE
Enhanced X-Ray Radiation Dose SR Storage	C-STORE
Enhanced XA Image Storage	C-STORE
Enhanced XRF Image Storage	C-STORE
General Audio Waveform Storage	C-STORE
General ECG Waveform Storage	C-STORE
Generic Implant Template Storage	C-STORE
Grayscale Softcopy Presentation State Storage	C-STORE
Hanging Protocol Storage	C-STORE
Hemodynamic Waveform Storage	C-STORE
Implant Assembly Template Storage	C-STORE
Implant Template Group Storage	C-STORE

Service	Command
Implantation Plan SR Document Storage	C-STORE
Intraocular Lens Calculations Storage	C-STORE
Intravascular Optical Coherence Tomography Image Storage - For Presentation	C-STORE
Intravascular Optical Coherence Tomography Image Storage - For Processing	C-STORE
Inventory	C-STORE
Keratometry Measurements Storage	C-STORE
Key Object Selection	C-STORE
Legacy Converted Enhanced CT Image Storage	C-STORE
Legacy Converted Enhanced MR Image Storage	C-STORE
Legacy Converted Enhanced PET Image Storage	C-STORE
Lensometry Measurements Storage	C-STORE
Macular Grid Thickness and Volume Report	C-STORE
Mammography CAD SR	C-STORE
Media Storage Directory Storage	C-STORE*
Microscopy Bulk Simple Annotations Storage	C-STORE
MR Image Storage	C-STORE
MR Spectroscopy Storage	C-STORE
Multi-channel Respiratory Waveform Storage	C-STORE
Multi-frame Grayscale Byte Secondary Capture Image Storage	C-STORE
Multi-frame Grayscale Word Secondary Capture Image Storage	C-STORE
Multi-frame Single Bit Secondary Capture Image Storage	C-STORE
Multi-frame True Color Secondary Capture Image Storage	C-STORE
Multiple Volume Rendering Volumetric Presentation State Storage	C-STORE
Nuclear Medicine Image Storage	C-STORE
Ophthalmic 16 bit Photography Image Storage	C-STORE
Ophthalmic 8 bit Photography Image Storage	C-STORE
Ophthalmic Axial Measurements Storage	C-STORE
Ophthalmic Optical Coherence Tomography B-scan Volume Analysis Storage	C-STORE
Ophthalmic Optical Coherence Tomography En Face Image Storage	C-STORE
Ophthalmic Tomography Image Storage	C-STORE

Service	Command
Ophthalmic Visual Field Static Perimetry Measurements Storage	C-STORE
Parametric Map Storage	C-STORE
Patient Radiation Dose SR Storage	C-STORE
Performed Imaging Agent Administration SR Storage	C-STORE
Planned Imaging Agent Administration SR Storage	C-STORE
Positron Emission Tomography Image Storage	C-STORE
Procedure Log	C-STORE
Protocol Approval Storage	C-STORE
Pseudo-Color Softcopy Presentation State Storage	C-STORE
Raw Data Storage	C-STORE
Real World Value Mapping Storage	C-STORE
Rendition Selection Document Real-Time Communication	C-STORE
Respiratory Waveform Storage	C-STORE
Robotic-Arm Radiation Record Storage	C-STORE
Robotic-Arm Radiation Storage	C-STORE
Routine Scalp Electroencephalogram Waveform Storage	C-STORE
RT Beams Delivery Instruction Storage	C-STORE
RT Beams Treatment Record Storage	C-STORE
RT Brachy Treatment Record Storage	C-STORE
RT Dose Storage	C-STORE
RT Image Storage	C-STORE
RT Ion Beams Treatment Record Storage	C-STORE
RT Ion Plan Storage	C-STORE
RT Patient Position Acquisition Instruction Storage	C-STORE
RT Physician Intent Storage	C-STORE
RT Plan Storage	C-STORE
RT Radiation Record Set Storage	C-STORE
RT Radiation Salvage Record Storage	C-STORE
RT Radiation Set Delivery Instruction Storage	C-STORE
RT Radiation Set Storage	C-STORE

Service	Command
RT Segment Annotation Storage	C-STORE
RT Structure Set Storage	C-STORE
RT Treatment Preparation Storage	C-STORE
RT Treatment Summary Record Storage	C-STORE
Secondary Capture Image Storage	C-STORE
Segmentation Storage	C-STORE
Segmented Volume Rendering Volumetric Presentation State Storage	C-STORE
Sleep Electroencephalogram Waveform Storage	C-STORE
Spatial Registration Storage	C-STORE
Spatial Fiducials Storage	C-STORE
Spectacle Prescription Report Storage	C-STORE
Standalone Overlay Storage	C-STORE
Standalone Curve Storage	C-STORE
Standalone Modality LUT Storage	C-STORE
Standalone VOI LUT Storage	C-STORE
Stereometric Relationship Storage	C-STORE
Subjective Refraction Measurements Storage	C-STORE
Surface Segmentation Storage	C-STORE
Tomotherapeutic Radiation Record Storage	C-STORE
Tomotherapeutic Radiation Storage	C-STORE
Ultrasound Image Storage	C-STORE
Ultrasound Multi-frame Image Storage	C-STORE
Variable Modality LUT Softcopy Presentation State Storage	C-STORE
Video Endoscopic Image Storage	C-STORE
Video Endoscopic Image Real-Time Communication	C-STORE
Video Microscopic Image Storage	C-STORE
Video Photographic Image Real-Time Communication	C-STORE
Video Photographic Image Storage	C-STORE
Visual Acuity Measurements Storage	C-STORE
VL Endoscopic Image Storage	C-STORE

Service	Command
VL Microscopic Image Storage	C-STORE
VL Photographic Image Storage	C-STORE
VL Slide-Coordinates Microscopic Image Storage	C-STORE
VL Whole Slide Microscopy Image Storage	C-STORE
Volume Rendering Volumetric Presentation State Storage	C-STORE
Wide Field Ophthalmic Photography 3D Coordinates Image Storage	C-STORE
Wide Field Ophthalmic Photography Stereographic Projection Image Storage	C-STORE
X-Ray Angiographic Image Storage	C-STORE
X-Ray Radiofluoroscopic Image Storage	C-STORE
X-Ray Radiation Dose SR Storage	C-STORE
X-Ray 3D Angiographic Image Storage	C-STORE
X-Ray 3D Craniofacial Image Storage	C-STORE
XA/XRF Grayscale Softcopy Presentation State Storage	C-STORE
XA Defined Procedure Protocol Storage	C-STORE
XA Performed Procedure Protocol Storage	C-STORE

NOTE: * Merge DICOM Toolkit defines a C-STORE command for the Media Storage Directory (DICOMDIR) service even though it does not formally exist In the DICOM Standard.

Finally, the DICOM file can be padded at the end with the Data Set Trailing Padding attribute (FFFC, FFFC) whose value is specified by the standard to have no significance.

2.4.2. File Sets

DICOM Files must be stored on removable media in a **DICOM File Set**. A DICOM file set is defined as a collection of DICOM files sharing a common naming space within which file ID's are unique (e.g., a file system partition). A DICOM File Set ID is a string of up to 16 characters that provides a name for the file set.

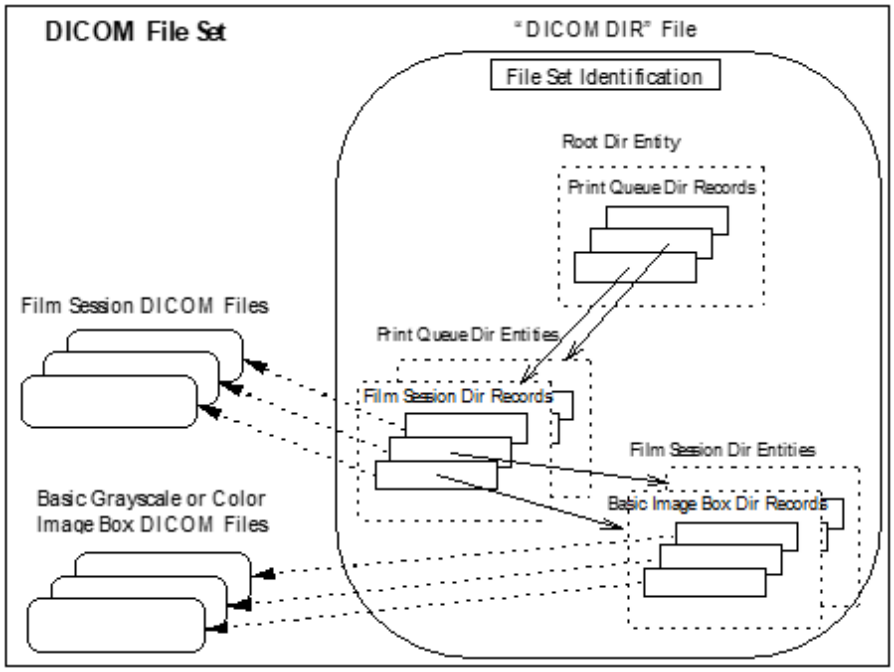
A **File ID** is a name given to a DICOM file that is mapped to each media format specification (in Part 12 of DICOM). A file ID consists of an ordered sequence of one to eight components, where each component is a string of one to eight characters. One can certainly imagine mapping such a file ID to a hierarchical file system, and this is done for several media formats in Part 12. It is important to note that DICOM states that no semantic relationship between DICOM files shall be conveyed by the contents or structure of file IDs (e.g., the hierarchy). This helps insure that DICOM files can be stored in a media format and file system independent manner.

The allowed characters in both a file ID and file set ID are a subset of the ASCII character set consisting of the uppercase characters (A-Z), the numerals (0-9), and the underscore (_).

2.4.3. The DICOMDIR

The **DICOM Directory File** or DICOMDIR is a special type of DICOM File. A single DICOMDIR must exist within each DICOM file set, and is always given the file ID "DICOMDIR". It is the DICOMDIR file that contains identifying information about the entire file set, and usually (dependent on the Application Profile) a directory of the file set's contents.

The figure below shows a graphical representation of a DICOMDIR file and its central role within a DICOM File Set.



a. The DICOMDIR Hierarchy

If the DICOMDIR file contains directory information, it is composed of a hierarchy of directory entities, with the top-most directory entity being the root directory entity. A **Directory Entity** is a grouping of semantically related directory records. A **Directory Record** identifies a DICOM File by summarizing key attributes and their values in the file and specifying the file ID of the corresponding file. The file ID can then be used, in the context of the native file system, to access the corresponding DICOM file. Each directory record can in turn point down the hierarchy to a semantically related directory entity.

Part 3 of the DICOM Standard specifies the allowed relationships between directory records in the section defining the Basic Directory IOD. We reproduce this table here (see the table below) for pedagogical reasons; but, you should refer to the DICOM Standard for the most up-to-date and accurate specification.

Table 2.6: Allowed Directory Entity

Directory Record Type	Record Types which may be included in the next lower-level Directory Entity
(Root Directory Entity)	PATIENT, HANGING PROTOCOL, PALETTE, IMPLANT, IMPLANT ASSY, IMPLANT GROUP, PRIVATE
PATIENT	STUDY, HL7 STRUC DOC, PRIVATE

Directory Record Type	Record Types which may be included in the next lower-level Directory Entity
STUDY	SERIES, PRIVATE
SERIES	IMAGE, RT DOSE, RT STRUCTURE SET, RT PLAN, RT TREAT RECORD, PRESENTATION, WAVEFORM, SR DOCUMENT, KEY OBJECT DOC, SPECTROSCOPY, RAW DATA, REGISTRATION, FIDUCIAL, ENCAP DOC, VALUE MAP, STEREOMETRIC, PLAN, MEASUREMENT, SURFACE, PRIVATE
IMAGE	PRIVATE
RT DOSE	PRIVATE
RT STRUCTURE SET	PRIVATE
RT PLAN	PRIVATE
RT TREAT RECORD	PRIVATE
PRESENTATION	PRIVATE
WAVEFORM	PRIVATE
SR DOCUMENT	PRIVATE
KEY OBJECT DOC	PRIVATE
SPECTROSCOPY	PRIVATE
RAW DATA	PRIVATE
REGISTRATION	PRIVATE
FIDUCIAL	PRIVATE
HANGING PROTOCOL	PRIVATE
ENCAP DOC	PRIVATE
HL7 STRUC DOC	PRIVATE
VALUE MAP	PRIVATE
STEREOMETRIC	PRIVATE
PALETTE	PRIVATE
IMPLANT	PRIVATE
IMPLANT ASSY	PRIVATE
IMPLANT GROUP	PRIVATE
PLAN	PRIVATE
MEASUREMENT	PRIVATE
SURFACE	PRIVATE
PRIVATE	PRIVATE, (any of the above as privately defined)

2.4.4. File Management Services and Roles

a. File Management Services

Part 10 of the DICOM Standard specifies a set of file management roles and services. There are five **DICOM File Services** that describe the entire set of DICOM file operation primitives:

Table 2.7: DICOM File Services

DICOM File Services	Description
M-WRITE	Create new files in a file set and assign them a file ID.
M-READ	Read existing files based on their file ID.
M-DELETE	Delete existing files based on their file ID.
M-INQUIRE FILE-SET	Inquire free space available for creating new files within a file set.
M-INQUIRE FILE	Inquire date and time of file creation (or last update if applicable) for any file within a file set.

The Merge DICOM Toolkit supplies families of functions that perform the first two file services. The Toolkit also implements enhanced read and write functionality for the creation and maintenance of DICOMDIR files and its hierarchy of directory entities and directory records. The remaining three file services are best implemented by the application entity through file system calls because they are file system dependent operations.

b. File Management Roles

DICOM Application Entities that perform file interchange functionality are in turn classified into three roles:

- File Set Creator (FSC) - Uses M-WRITE operations to create a DICOMDIR file and one or more DICOM files.
- File Set Reader (FSR) - Uses M-READ operations to access one or more files in a DICOM file set. An FSR shall not modify any files of the file set (including the DICOMDIR file).
- File Set Updater (FSU) - Performs M-READ, M-WRITE, and M-DELETE operations. It reads, but shall not modify the content of any DICOM files other than the DICOMDIR file. It may create additional files by means of an M-WRITE or delete existing files by means of an M-DELETE.

The concept of these roles is used within the DICOM conformance statement of an application entity that supports media interchange to express the capabilities of the implementation more precisely. Conforming applications shall support one of the capability sets specified in the table below. DICOM conformance is described in greater detail in the next section.

Table 2.8: Media Application Operations and Roles

Media Roles	M-WRITE	M-READ	M-DELETE	M-INQUIRE FILE-SET	M-INQUIRE FILE
FSC	Mandatory	not required	not required	Mandatory	Mandatory
FSR	not required	Mandatory	not required	not required	Mandatory

Media Roles	M-WRITE	M-READ	M-DELETE	M-INQUIRE FILE-SET	M-INQUIRE FILE
FSC+FSR	Mandatory	Mandatory	not required	Mandatory	Mandatory
FSU	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory
FSU+FSC	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory
FSU+FSR	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory
FSU+FSC+FSR	Mandatory	Mandatory	Mandatory	Mandatory	Mandatory

2.5. Conformance

Part 2 of DICOM discusses conformance and is important to any AE developer. For an application to be DICOM conformant it must:

- meet the minimum general conformance requirements specified in Part 2 and service specific conformance requirements specified in Part 4 (Network Services), and/or Parts 10 and 11 (Media Services); and
- have a published DICOM conformance statement detailing the above conformance and any optional extensions.

Conformance also applies to aspects of the communications protocol that are managed by the DICOM Toolkit. Most parameters are configurable by your application.

Conformance Statement templates in each of the Sample Application Guides also provide guidance in preparing your conformance statement for your application.

Part 2 also deals with private extensions to the DICOM Standard by defining **Standard Extended Services**. Standard Extended Services give your application a little more flexibility, by allowing you to add private attributes as long as they are of value type 3 (optional) and are documented in the conformance statement.

DICOM also allows you to define your own **Specialized** and **Private Services**. These should be avoided by most applications since they are non-standard, add complexity to your application, and limit interoperability.

If you are significantly extending services or creating your own private services, you may need the Merge DICOM Toolkit Extended Toolkit to assist in defining these services so that they can be supported by the toolkit.

Chapter 3. Using Merge DICOM Toolkit

You can use the Merge DICOM Toolkit 'out of the box' by using its supplied utility programs and sample applications. In this section we discuss how to configure the toolkit and to use the utility programs. Use of the sample applications is described in the sample application guides. Later, we discuss how to develop your own DICOM applications using the Merge DICOM Toolkit library.

3.1. Configuration

Merge DICOM Toolkit is highly configurable, and understanding its configuration files is critical to using the library effectively.

Related parameters are grouped into sections in a configuration file as follows:

```
[SECTION_1]
   PARAMETER_1 = value1
   PARAMETER_2 = value2
[SECTION_2]
   PARAMETER_3 = value3
...
```

Related sections are grouped into one of four configuration files:

- initialization file
- application profile
- system profile
- service profile

Each of these configuration files is discussed separately below. Only the key configurable parameters are summarized in this document.

3.1.1. Initialization File

The Merge DICOM Toolkit Initialization File (usually called `merge.ini`) provides the DICOM Toolkit with its top-level configuration. It specifies the location of the other three configuration files, along with message and error logging characteristics.

The method `MC.initialize(inifile, license)` can be used to assign the path where the merge.ini file is located. You can also set the `MERGE_INI` environmentvariable to point to the Merge Initialization File. This variable can be set within a command shell; for example:

In Unix C-shell:

```
setenv MERGE_INI /users/mc3adv/merge.ini
```

In Unix Bourne, Korn, or Bash shell:

```
MERGE_INI=/users/mc3adv/merge.ini; export MERGE_INI
```

In DOS command shell:

```
set MERGE_INI=\mc3adv\merge.ini
```

The initialization file contains one [MergeCOM3] section that points to the location of the other three Merge DICOM Toolkit initialization files, specifies characteristics of the message/error log kept by the DICOM Toolkit library, turns particular types of logging on and off, and specifies where the messages are logged (file, screen, both, or neither). In most cases the INFO, WARNING, and ERROR messages will be sufficient. The Tn_MESSAGE settings (where *n* is an integer between 1 and 9) turns on lower-level protocol tracing capabilities. These capabilities can prove useful when running into difficulties communicating with other implementations of DICOM over a network and can be used by Merge service engineers in diagnosing lower-level network problems.

3.2. Message Logging

Merge DICOM Toolkit supplies a message logging facility whereby three primary classes of messages can be logged to a specified file and/or standard output:

- Errors
- Warnings
- Info

Error messages include unrecoverable errors, such as "association aborted", or "failure to connect to remote application". Other error messages may be catastrophic but it is left to the application to determine whether or not to abort an association, such as an "invalid attribute value" or "missing attribute value" in a DICOM message.

Warnings are meant to alert toolkit users to unusual conditions, such as missing parameters that are defaulted or attributes having values that are not one of the defined terms in the standard.

Info messages include high-level messages describing the opening of associations and exchanging of messages over open associations.

As discussed earlier, other more detailed logging can be obtained by using the T1_MESSAGE through T9_MESSAGE logging levels. For example, the T5_MESSAGE logging level can be used to log the results of an **MCattributeSet.validate(level)** call.

The trace logging levels are intended strictly for debugging purposes. If left on, they can seriously degrade toolkit performance. In particular, the T2, T7 and T9 levels should be turned off in normal operation.

An excerpt from a Merge DICOM Toolkit message log file is included below that contains all three classes of messages: errors, warnings, and informational.

Message Log Example:

```
...
03-29 21:14:54.77 MC3 W: (0010,1010): Value from stream had problem:
03-29 21:14:54.78 MC3 W: |   Invalid value for this tag's VR
03-29 21:14:56.41 MC3(Read_PDU_Head) E: Error on Read_Transport call
03-29 21:14:56.41 MC3(MCI_nextPDUtype) E: Error on Read_PDU_Head call
03-29 21:14:56.41 MC3(Transport_Conn_Closed_Event) E: Transport
unexpectedly closed
03-29 21:14:56.41 MC3(MCI_ReadNextPDV) I: DUL_read_pdvs error: UL
Provider aborted the association
03-29 21:14:56.41 MC3 E: (0000,0000): Error during
MC_Stream_To_Message:
```

```
03-29 21:14:56.41 MC3 E: | Callback cannot comply

03-29 21:14:56.41 MC3(MC_Read_Message) E: Network connection
unexpectedly shut down

...
```

On many DICOM Toolkit computing platforms, additional information is logged, such as process and thread id numbers identifying where the message was generated.

3.3. Utility Programs

The Merge DICOM Toolkit supplies several useful utility programs. These utilities can be used to help you validate your own implementations and better understand the standard.

All these utilities use the Merge DICOM Toolkit Library and require that you set your `MERGE_INI` environmental variable to point to the proper configuration files (as described earlier).

3.3.1. mc3comp

The mc3comp utility can be used to compare the differences between two DICOM objects. The objects can be encoded in either the DICOM file or "stream" format and do not have to be encoded in the same format. The utility will output differences in tags between the messages taking into account differences in byte ordering and encoding. The syntax for the utility is the following:

```
mc3comp [-t1 <syntax> -t2 <syntax>] [-e file] [-o -m1 -m2] file1 file2


-t1 <syntax>Optional specify transfer syntax of 'file1'message, where
<syntax> = 'il' for implicit little endian (default),
           'el' for explicit little endian,
           'eb' for explicit big endian.
-t2 <syntax>Optional specify transfer syntax of 'file2' message,
where
<syntax> = 'il' for implicit little endian (default),
           'el' for explicit little endian,
           'eb' for explicit big endian.
-e <file>Optional exception file of all tags to ignore in comparison.
-oCompare OB/OW/OF (e.g.,binary pixel) data.
-m1Compare 'file1' in DICOM-3 file format.
-m2Compare 'file2' in DICOM-3 file format.
-hShow these options.
file1DICOM SOP Instance (message) file
file2Another DICOM SOP Instance (message) file


Example:mc3comp -t1 il -m2 -o 1.img 1.dcm
```

3.3.2. mc3conv

The `mc3conv` utility can be used to convert a DICOM object between various transfer syntaxes and formats. The utility will read an input file and then write the output file in the transfer syntax specified in the command line. The utility can also convert between DICOM "stream" format and the DICOM file format. The syntax for the `mc3conv` utility is the following:

```
mc3conv input_file output_file [-t <syntax>] [-p] [-m] [-x] [-s
<syntax> [-tag <tag> <"new value">]


input_fileDICOM SOP Instance (message) file.
output_fileOutput DICOM SOP Instance (message) file.
-tSpecify transfer syntax for 'output_file', where
<syntax> = il' for implicit little endian (default)
                         'el' for explicit little endian
                         'eb' for explicit big endian
                         'ib' for implicit big endian
                         'jb' for jpeg baseline
                         'je' for jpeg extended 2_4
                         'jl' for jpeg lossless hier 14
                         'j2lo' for jpeg 2000 lossless only
                         'j2' for jpeg 2000
                         'rle' for rle


-mSpecify format of 'output_file' to be DICOM-3 media (Part 10)
               format.
-sSpecify transfer syntax for 'input_file'.
-pJust extract the pixel data from 'input_file' into
'output_file'.If multiframe and encapsulated, '_x'
is appended to 'output_file' for each frame.
-tagChange value for this tag in 'output_file', where
<tag>       = the tag that is to be changed in hex 0x...
<new value> = the value for the tag in quotes,
multi values separated as "val1\val2".
-xSpecify format of 'output_file' to be XML format.
-hShow these options.


Example: mc3conv in.img out.dcm -t el -m
```

3.3.3. mc3echo

The `mc3echo` utility validates application level communication between two DICOM AEs. An echo test is the equivalent of a network 'ping' operation, but at the DICOM application level rather than the TCP/IP transport level.

All server (SCP) applications built with the DICOM Toolkit also have built-in support of the Verification Service Class and the C-ECHO command.

The command syntax follows:

```
mc3echo [-c count] [-r remote_host] [-l local_app_title] [-p
              remote_port] remote_app_title


-c countInteger number specifying the number of echoes to send to the
              remote host. If -c is not specified, one echo will be
              performed.

-r remote_hostHost name of the remote computer If -r is not specified,
              the default value for remote_host is configured in the
              Application Profile.

-l local_app_titleApplication title of this program.


              If -l is not specified, the default value

              For local_app_title is MERGE_ECHO_SCU


-p remote_portPort number the remote computer is listening on. If -p
              is not specified, the default value for remote_host is
              configured in the Application Profile.
```

3.3.4. mc3list

`mc3list` displays the contents of binary DICOM message files in an easy to read manner. The message files could have been generated by mc3file (see below) or written out by your application.

`mc3list` is a useful educational tool as well as a tool that can be used for off-line display of the DICOM messages your application generates or receives.

The command syntax follows:

```
mc3list <filename> [-t <syntax>] [-m]


filename  Filename containing message to display

-t        Specify transfer syntax of message, where syntax is:

                        "il" (implicit little endian),

                        "el" (explicit little endian), or

                        "eb" (explicit big endian)

-m        Optional display a DICOM file object
```

If the DICOM service and/or command cannot be found in the message file, a warning will be displayed, but the message will still be listed.

The default transfer syntax is implicit little endian (the DICOM default transfer syntax). If the transfer syntax is incorrectly specified, the message will not be displayed correctly.

3.3.5. mc3valid

The `mc3valid` utility validates binary message files according to the DICOM standard and notifies you of missing attributes, improper data types, illegal values, and other problems with a message. `mc3valid` is a powerful educational and validation tool that can be used for the off-line validation of the DICOM messages your application generates or receives.

The command syntax follows:

```
mc3valid <filename> [-e|-w|-i] [-s <serv> -c <cmd>] [-p] [-q] [-t
<syntax>]


<filename> Filename containing message to validate
-e         Display error messages only (optional)
-w         Display error and warning messages (optional,
           default)
-I         Display informational, error, and warning messages
           (optional)
-s <serv>  Force the message to be validated against service name
           "serv", used along with '-c' (optional)
-c <cmd>   Force the message to be validated against command name
           "cmd", used along with '-s' (optional)
-q         Disable prompting for correct service-command pairs
           (optional)
-p         Use message template to validate message against
           (optional, maintained for backward compatibility only)
-t         Specify transfer syntax of message, where
           syntax = "il" (implicit little endian)
                  = "el" (explicit little endian)
                  = "eb" (explicit big endian)
```

This command validates the specified message file; printing errors, warnings, and information generated to standard output. The user can force the message to be validated against a specified DICOM service-command pair if the message does not already contain this information.

If the service-command pair is not contained in the message, the program will list the possible service-command pairs and the user can select one of them. When using this program with a batch file, this option can be shut off with the -q flag.

The default transfer syntax is implicit little endian (the DICOM default transfer syntax). If the transfer syntax is incorrectly specified, the message cannot be validated.

Limitations

While `mc3valid's` message validation is quite comprehensive, it does have limitations. The DICOM Standard should always be considered the final authority.

3.3.6. mc3file

Sample DICOM messages can be generated with the `mc3file` utility. You specify the service, command, and transfer syntax and `mc3file` generates a 'reasonable' sample message that is written to a binary file. The contents of this file are generated in DICOM file format or in exactly the format as the message would be streamed over the network.

The program fills in default values for all the required attributes within the message. You can also use this utility to generate its own configuration file, which you can then modify to specify your own values for attributes in generated messages.

These generated messages are purely meant as 'useful' examples that can be used to test message exchange or give the application developer a feel for the structure of DICOM messages. They are not intended to represent real world medical data.

The messages generated can be validated or listed with the `mc3list` and `mc3valid` utilities. The command syntax for `mc3file` is the following:

```
mc3file <serv> <cmd> <num> [-g <file>] [-c <file>] [-l] [-m] [-q] [-t
         <syntax>] [-f

<file>]


<serv>These two options are always used together.


<cmd>They specify the service name and command for the message to be
         generated. These names can be either upper or lower case. If
         the exact names for a service command pair are not known, the
         -l option can be used instead to specify the service name and
         command. If the service name and command are improperly
         specified, mc3file will act as if the -l option was used and
         ask the user to input the correct service name and command.

<num>This option specifies the number of message files to be generated
         by mc3file. If the -g option is used, this option is not
         needed on the command line. If the -c option is used, mc3file
         assumes the number is 1, although a higher number can be
         specified on the command line. mc3file will vary any fields
         that have a value representation of time when multiple files
         are generated, although when the -c option is used, the
         utility will use the time fields as specified in the
         configuration file. Thus, multiple message files generated
         with the -c option are identical.
```

```
-g <file>This option causes mc3file to generate an ASCII
        configuration file. The file contains a listing of all the
        valid attributes for the specified message. The utility also
        adds sequences contained in the message along with their
        attributes. Each attribute in the file contains the tag,
        value representation, and the default value MC3File uses for
        the attribute. If a given attribute has more than one value,
        the character "\" is used to delimit the values. A default
        value listed as "NULL" means the attribute is set to NULL. If
        the filename specified already exists, it will be written
        over my MC3File. The configuration file can be modified and
        reloaded into MC3File with the -c option to generate a DICOM
        message.

-c <file>This option reads in a configuration file previously
        generated by mc3file. The service name and command for the
        message need not be specified on the command line because
        they are contained in <filename>. Because multiple files
        generated with this option are identical, mc3file assume
        only one file should be generated. This assumption can be
        overridden by specifying a number on the command line.

-lThis option lists all the service command pairs supported by
        mc3file. When generating a message, this option can be used
        instead of explicitly specifying the service name and
        command on the command line. When specified alone in the
        command line, the complete list of pairs is printed out
        without pausing.

-mThis option allows the user to generate a DICOM file. When
        generating the file object, mc3file encodes the File Meta
        Information.

-qThis option prevents mc3file from prompting the user for correct
        service command pairs. It is a useful option when running
        the program from a batch file.

-t <syntax>This option specifies the transfer syntax the DICOM
        message generated is stored in. The default transfer syntax
        is implicit little endian. The possible values for <syntax>
        are "il" for implicit little endian, "el" for explicit
        little endian, and "eb" for explicit big endian.

-f <file>This option allows the user to specify the first eight
        characters of the names of the DICOM message files being
        generated. mc3file will then append a unique count to the
        end of the filename for each message being generated. The
        default value is "file" when creating a DICOM file and
        "message" when creating the format that DICOM messages send
        over a network
```

Chapter 4. Developing DICOM Applications

The Merge DICOM Toolkit Application Programming Interface (API) provides simple yet powerful DICOM functionality. Function calls are provided that open associations with remote servers, wait for associations from remote clients, and deal with DICOM message exchange over an open association. Functions are also provided for the creation and reading of DICOM files and the creation, maintenance, and navigation of DICOMDIRs. DICOM Toolkit features include message validation against the DICOM Standard, support of sequences of items, methods for flexible handling of Pixel Data, and support of Private Attributes.

This section of the User's Manual attempts to present the highlights of the Merge DICOM Toolkit API in a logical manner as it might be used in real DICOM applications. The function calls are presented in the context of example ANSI-C source code snippets, and alternative approaches are presented that trade off certain features for the benefits of increased performance.

Most of the discussions that follow pertain both to networking and media interchange applications; only 4.3. Association Management (Network Only) on page 48, 4.4. Negotiated Transfer Syntaxes (Network Only) on page 50 and 4.7. Message Exchange (Network Only) on page 62 sections are networking specific. Section 4.10. DICOM Files on page 72 is media interchange specific.

4.1. Library Initialization

Your first call to the Merge DICOM Toolkit Library must always be the **`MC.initialize(inifile, license)`** function. This function specifies how and when you wish to initialize the library with the contents of its configuration files, data dictionary, and message info files.

Almost all typical applications will initialize themselves from configuration files, and will make use of the binary dictionary and message info files in building message objects.

Configurable parameters can be modified by your application after library initialization, at runtime, by using the **`MCconfig.set()`** methods. The tooltkit allows you to initialize an internal system exception handler and add the user-defined exception handler, which will be called in case of severe system error or signal.

Function calls to Merge DICOM Toolkit throws an exception if any error occurs. The error code might be checked against an exception ID number. Any value other than `MCstatus.MC_NORMAL_COMPLETION` signifies an error.

4.2. Registering Your Application

Before performing any network or media activity, your application must register its **DICOM Application Title** with the Merge DICOM Toolkit to refer to this particular AE in subsequent function calls.

This DICOM Application Title is equivalent to the DICOM Application Entity Title defined earlier. If your application is a server, this application title must be made known to any client application that wishes to connect to you. If your application is a client, your application title may need to be made known to any server you wish to connect to, depending on whether the server is configured to act as an server (SCP) only to particular clients for security reasons.

For example, if your application title is "ACME_Query_SCP", you would register with the toolkit as follows:

app = `MCapplication.getApplication('ACME_Query_SCP')`

If you wish to disable your application and free up its resources to the system you should call `app.dispose()` method.

Current and potentially future DICOM service classes assume that Application Entity Titles on a DICOM network are unique. For instance, the retrieve portion of the Query/Retrieve service class specifies that an image be moved to a specific Application Entity Title (and not to a specific hostname and listen port). If two identical Application Entity Titles existed on a network, a server application can only be configured to move images to one of these applications. For this reason, the DICOM Application Entity Title for your applications should be configurable.

4.3. Association Management (Network Only)

Once you have registered one or more networking applications, you will probably want to initiate an association if you are a client, or wait for an association if you are a server.

To initiate an association as a client, you make an `MCassociation.requestAssociation()` call. You specify the parameters of Remote Application Title of the server you wish to connect to. If the association is accepted, the function returns normally, with an Association object that is used in future calls. When you are done making DICOM service requests (sending and receiving messages) over the association, you should release the association with an `MCassociation.release()` call.

Client Side Example:

```
#
# Creates a local DICOM application object
#
application = MCapplication.getApplication(aetitle)


#
# Creates a remote DICOM application object
#
remoteApplication = MCremoteApplication('remoteAETitle',
'remoteHost', 'remotePort', MCproposedContextList('service'))


#
# Request DICOM association from remote DICOM application
#
application.requestAssociation(remoteApplication)
#
# Reads DICOM Media file
#
dcm = MCfile()
```

```
dcm.readP10File(f)

#
# Reads Transfer Syntax and DICOM Service
#
ts = dcm.getTransferSyntax()
svc = dcm.getService()

#
# Creates DICOM message from DICOM file and sends C-STORE request
#
msg = MCdimseMessage.fromFile(dcm)
msg.setServiceCommand(svc, MCcommand.C_STORE_RQ)
msg.setTransferSyntax(ts)

response = assoc.sendRequestMessage(msg, svc)
```

The SCP server application starts a listener for DICOM association requests directed to this application using the `startListening(address, MCproposedContextList('service'), MChandler)` method, where `MChandler` is an object instance which implements the `handle(association)` method and that will handle the received associations. The SCP server application detects when the client has requested association. Upon successful connection, the association for the incoming connection is passed to `MChandler.handle(association)` to actually process the connection. After completion, a typical server application processes the incoming connection and returns back to waiting for the next incoming association.

Server Side Example of `MChandler.handle(association)` method:

```
def handle(self, obj):

    association = obj
    if not association.isActive():
       return

    try:
       association.accept()
       while association.isActive():
           rd = association.read(TIMEOUT)
           if (rd == None):
               continue

           status = rd.getStatus()
```

```
            if(status == MCstatus.MC_TIMEOUT):
                continue
            elif(status == MCstatus.MC_ASSOCIATION_ABORTED) or \
                 (status == MCstatus.MC_ASSOCIATION_CLOSED):
                break
            elif(rd.getMessage() == None):
                break

             msg = rd.getMessage()
             if(msg.getCommand() == MCcommand.C_STORE_RQ):
                 processCStore( association, msg)
             elif(msg.getCommand() == MCcommand.C_FIND_RQ):
                 processCFind( association, msg)
             elif(msg.getCommand() == MCcommand.C_GET_RQ):
                 processCGet( association, msg)
             elif(msg.getCommand() == MCcommand.C_MOVE_RQ):
                 processCMove( association, msg)
             elif(msg.getCommand() == MCcommand.N_ACTION_RQ):
                 processNAction( association, msg)
             else:
                 raise MCexception(MCexception.OPERATION_NOT_ALLOWED)
        except Exception as ex:
            exception = MCexception.create(ex)
        finally:
            if association != None:
               try:
                  association.abort()
               except:
                  pass
```

4.4. Negotiated Transfer Syntaxes (Network Only)

Merge DICOM Toolkit supports all currently approved standard and encapsulated DICOM transfer syntaxes. Encapsulated transfer syntaxes require compression of the pixel data contained in the message. These messages can be sent and received by the toolkit, although the toolkit will not do the actual compression and decompression. Encoding of this pixel data is discussed below.

For DICOM Toolkit users, the toolkit allows for the negotiation of more than one transfer syntax for a given DICOM service. This functionality is of most use for applications supporting encapsulated transfer syntaxes. This functionality may be disabled by use of the `ACCEPT_MULTIPLE_PRES_CONTEXTS` configuration value. In order to understand how it is implemented, a more in depth description of DICOM association negotiation is required.

During association negotiation a client (SCU) application will propose a set of presentation contexts over which DICOM communication can take place. Each presentation context consists of an abstract syntax (DICOM service) and a set of transfer syntaxes that the client (SCU) understands. The server (SCP) will typically accept a presentation context if it supports the abstract syntax and one of the proposed transfer syntaxes.

As previously discussed, the abstract and transfer syntaxes supported by a server (SCP) are defined through a service list contained in the Merge DICOM Toolkit Application Profile. When support within a server (SCP) is limited to the three non-encapsulated DICOM transfer syntaxes, the toolkit will transparently handle the use of multiple presentation contexts for a DICOM service. However, when encapsulated DICOM transfer syntaxes are used, the server (SCP) must be able to determine the transfer syntax of messages it receives so that it can properly parse the pixel data contained in them.

4.4.1. Transfer Syntax Lists for SCUs

The presentation contexts supported for client (SCU) applications using Merge DICOM Toolkit are also defined through the Merge DICOM Toolkit Application Profile. The following is a typical client (SCU) configuration:

```
[Acme_Store_SCP]
PORT_NUMBER          = 104
HOST_NAME            = acme_sun1
SERVICE_LIST         = Storage_Service_List


[Storage_Service_List]
SERVICES_SUPPORTED   = 1  # Number of Services
SERVICE_1            = STANDARD_CT
```

In this case, the client (SCU) would propose the CT Image Storage service in a single presentation context. The transfer syntaxes for each service are the three standard (non-encapsulated) DICOM transfer syntaxes.

The following example is the configuration for a client (SCU) that supports more than one presentation context for a service:

```
[Acme_Store_SCP]
PORT_NUMBER          = 104
HOST_NAME            = acme_sun1
SERVICE_LIST         = Storage_Service_List


[Storage_Service_List]
SERVICES_SUPPORTED   = 2  # Number of Services
```

```
SERVICE_1            = STANDARD_CT
SYNTAX_LIST_1        = CT_Syntax_List_1
SERVICE_2            = STANDARD_CT
SYNTAX_LIST_2        = CT_Syntax_List_2

[CT_Syntax_List_1]
SYNTAXES_SUPPORTED   = 1  # Number of Syntaxes
SYNTAX_1             = JPEG_BASELINE

[CT_Syntax_List_2]
SYNTAXES_SUPPORTED   = 1  # Number of Syntaxes
SYNTAX_1             = IMPLICIT_LITTLE_ENDIAN
```

If a server (SCP) accepts both of these presentation contexts, the client (SCU) must use the **`MCattributeSet.setTransferSyntax(ts)`** method to specify which presentation context to send a message over.

4.4.2. Transfer Syntax Lists for SCPs

Server (SCP) applications are configured differently than client (SCU) applications. An SCP should include all of the transfer syntaxes a service supports in a single transfer syntax list. If more than one transfer syntax list is used for a service, server (SCP) applications will only support the transfer syntaxes contained in the first transfer syntax list. The following is an example configuration for a server (SCP):

```
[Storage_Service_List]
SERVICES_SUPPORTED   = 1  # Number of Services
SERVICE_1            = STANDARD_CT
SYNTAX_LIST_1        = CT_Syntax_List_SCP

[CT_Syntax_List_SCP]
SYNTAXES_SUPPORTED   = 4  # Number of Syntaxes
SYNTAX_1             = JPEG_BASELINE
SYNTAX_2             = EXPLICIT_LITTLE_ENDIAN
SYNTAX_3             = IMPLICIT_LITTLE_ENDIAN
SYNTAX_4             = EXPLICIT_BIG_ENDIAN
```

As discussed previously, for server (SCP) applications, the order in which transfer syntaxes are specified in a transfer syntax list dictates the priority Merge DICOM Toolkit places on them during association negotiation. In this case, Merge DICOM Toolkit would select `JPEG_BASELINE` if

proposed, followed by `EXPLICIT_LITTLE_ENDIAN`, `IMPLICIT_LITTLE_ENDIAN`, and `EXPLICIT_BIG_ENDIAN`.

Network message exchange is discussed further in one of the following sections.

4.5. Dynamic Service Lists

In addition to defining service lists in the Application Profile, Merge DICOM Toolkit has mechanisms to define service lists and transfer syntax lists at run-time. A number of functions exist to create transfer syntaxes and service lists in various formats. The following example shows how to create two transfer syntax lists and a service list:

```
#
# Creates Transfer Syntax list
#
tsl = MCtransferSyntaxList('tslJPEG', [MCtransferSyntax.JPEG_2000])
#
# Creates Proposed Context
#
pc = MCproposedContext(MCsopClass.getByName(MCservices.STANDARD_CT),
tsl)


#
# Creates Proposed Context list
#
pcl = MCproposedContextList("PCL", [pc])
```

In this example, `MCtransferSyntaxList` is used to create a transfer syntax list. This object is passed an array of transfer syntaxes that are placed in the list and the user specifies a name for the syntax list. Similar to the creation of syntax lists in the application profile, the order in which transfer syntaxes are defined in the list dictates the priority Merge DICOM Toolkit places on the transfer syntaxes when negotiating an association.

The `MCproposedContextList` object can then be used to create a new service list consisting of created `MCproposedContext` services.

4.6. Message Objects

Merge DICOM Toolkit supplies several types of **objects**: application objects, association objects, message objects, file objects, and item objects. Objects provide a convenient way for the toolkit to encapsulate related data while hiding unnecessary details from the application developer.

A majority of the functionality supplied with the DICOM Toolkit deals with building, parsing, validation, and exchange of DICOM messages, files, and items. Your applications deal with network messages in Merge DICOM Toolkit as message objects, and DICOM files as file objects. Item

objects are used for attributes that are of VR Sequence of Items (SQ) within both messages and files.

This section deals with message objects, but many of these functions are polymorphic and also work on file and item objects. These polymorphic functions will be called out in this section. Additional functions that are particular to file objects or item objects will be described in later sections.

Private attributes in both message and file objects are handled in ways similar to those discussed in this section, but will also be described later in this document.

4.6.1. Building Messages

Before you can build a message, your application must create a message object using the constructor of **`MCdimseMessage`** class inherited from the **`MCattributeSet`** base class. Your application must also specify a Service and Command name using `MCdimseMessage.setServiceCommand(service, command)` method. The DICOM Toolkit library uses these parameters to reference the proper message info file along with the data dictionary and builds an unpopulated message object instance for your application to fill in. This message object contains empty attributes. A Message ID is returned to your application that identifies this message object.

Once you have an open message object, use the `McdimseMessage.addValue(tag, value)` and `MCdimseMessage.setValue() (tag, value)` methods to build your message. The methods perform the necessary type conversion from any reasonable ANSI-C data type to any compatible DICOM value representation.

For instance, loss of precision is possible when `MCdimseMessage.setValue()` is called to set the value of a float or a double attribute from a string. Let's assume the value of the attribute (of VR=FL) is 123.456789. Internally, the toolkit converts the value to string using the %g format specification. The returned result is the "123.457" string (the rounded value with the default precision).

If a type conversion is not reasonable (e.g., from `short int` to LT), then exception will be thrown with `MC_INCOMPATIBLE_VR` id. Also, other exceptions will be thrown if the conversion was reasonable but the value stored in the variable made the conversion impossible (exceptions with `MC_INVALID_VALUE_FOR_VR`, `MC_VALUE_OUT_OF_RANGE`...ids).

Table 4.1: Acceptable Set Value/VR combinations

Set type	Function may be used to set values of attributes with these Value Representations
`Int`	AT, DS, FD, FL, IS, SS, SL, SV, US, UL, UV, SQ
`Float`	DS, FD, FL, IS, SS, SL, SV, US, UL, UV, SQ
`Str`	AS, AT, CS, DA, DS, DT, FD, FL, IS, LO, LT, PN, SH, SS, SL, SV, ST, TM, UI, US, UL, UV, UC, UR, UT, SQ
`Bytes`	OB, OW, OL, OV, OF, OD, UT, UNKNOWN_VR

Large binary (bulk) values, like pixel data, can be read or set by using io.IOBase streams or by simple byte arrays. The toolkit stores bulk data in either memory or temporary files depending on the value of LARGE_DATA_STORE and LARGE_DATA_SIZE configuration items in mergecom.pro. Alternatively, custom value storage may be used for handling such values. See the description of the `MCapplication.registerProvider(MCIstorageProvider, tagnumber)` method for details on custom value storage.

If your application runs on a resource rich system, you should set `LARGE_DATA_STORE` to the value `MEM` in the Service Profile, and Merge DICOM Toolkit will keep the Pixel Data values in the message object stored in memory rather than using temporary files to improve performance.

4.6.2. Parsing Messages

When your AE receives a DICOM message, it will most often need to examine the values contained in the message attributes to perform an action (e.g., store an image, print a film, change state...). If your application is a server, the message conveys the operation your server should perform and the data associated with the operation. If your application is a client, the message may be a response message from a server on the network resulting from a previous request message to that same server.

Once you have received a message object, use the `McdimseMessage.getValue()` method to parse your message.

The `McdimseMessage.getValue()` family of functions also operate on DICOM file objects and item objects, since both of these objects are also constructed of DICOM attributes.

This family of functions can be broken into four types based on their functionality as specified in the table below.

Table 4.2: The Get Value Family of Methods

Function Type	Description
`getValueCount()`	Returns the number of values assigned to an attribute in a message object. Multi-valued attributes can have more than one value assigned to them.
`getValueLength()`	Returns the length of attribute's value in bytes.
`getValue()`	Gets the first (and possibly only) value of an attribute in a message object. There are eleven functions of this type, the one you use depends on the data type of the variable to which you are assigning to the attribute's value (e.g., string, short int, long int, float, ...).
`getValues()`	Gets an array containing each value of the attribute. The element type of the array depends on the value representation of the attribute. The length of the returned array is equal to the number of values of the attribute, except for bulk type attributes (OB, OW, OL, OD, OF, UN) for which the length of the array is equal to the attribute's length. For bulk attributes the return value is the same as the return value of the getValue() method. If the attribute has no values an array of zero length is returned.

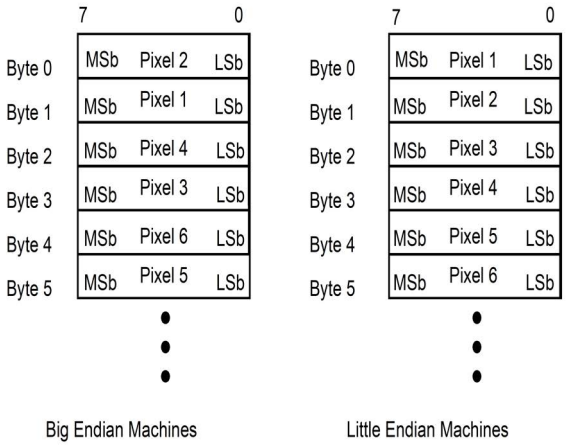
NOTE: The same acceptable conversions applies for the private DICOM attributes.

4.6.3. 8-bit Pixel Data

For DICOM's Implicit VR Little Endian transfer syntax, the pixel data attribute's (`7fe0,0010`) VR is specified as being OW (independent of what the bits allocated and bits stored attributes are set to). To reduce confusion, Merge DICOM Toolkit sets the VR of pixel data for the other non-encapsulated transfer syntaxes to OW.

When retrieving or setting pixel data with the `MCdimseMessage.getValue()` and `MCdimseMessage.setValue()` methods, the toolkit assumes that the OW pixel data is encoded in

the host system's native endian format as defined by DICOM. The figure below describes how 8-bit pixel data is encoded in an OW buffer for both big and little endian formats.



The DICOM standard specifies that the first pixel byte should be set to the *least significant byte* of the OW value. The next pixel byte should be set to the *most significant byte* of the OW value.

4.6.4. Encapsulated Pixel Data

Merge DICOM Toolkit supports handling of single frame and multi-frame pixel data in encapsulated transfer syntaxes, dealing with it in the same manner as standard pixel data by using the following `MCattributeSet` calls:

```
getEncapsulatedFrame()
setEncapsulatedFrame()
getFrame()
```

Merge DICOM Toolkit will encode supplied data in an encapsulated format and generate the basic offset table. The data of basic offset table could be retrieved using `MCattributeSet` call:

```
getOffsetTable()
```

Merge DICOM Toolkit will provide data without the encapsulation delimiters. The data can be compressed or decompressed using registered compression and decompression callbacks. Registration of compression/decompression callbacks is described later in this document. Compression libraries are also included on several platforms, including Windows, Sun Solaris and Linux.

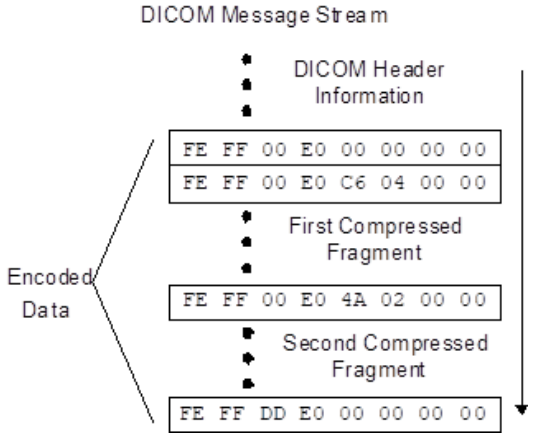
An example of encapsulated pixel data is illustrated in the table below.

Table 4.3: Sample Encapsulated Pixel Data

Pixel Data Element									
Basic Offset Table with NO Item Value		First Fragment (Single Frame) of Pixel Data			Second Fragment (Single Frame) of Pixel Data			Sequence Delimiter Item	
Item Tag	Item Length	Item Tag	Item Length	Item Value	Item Tag	Item Length	Item Value	Sequence Delim. Tag	Item Length

(FFFE, E000)	0000 0000H	(FFFE, E000)	0000 04C6H	Compressed Fragment	(FFFE, E000)	0000 024AH	Compressed Fragment	(FFFE, E0DD)	0000 0000H
4 bytes	4 bytes	4 bytes	4 bytes	04C6H bytes	4 bytes	4 bytes	024A H bytes	4 bytes	4 bytes

As specified by the DICOM standard, the various elements shown in the table above, excluding the compressed pixel data fragments, are encoded in little endian format. The compressed pixel data fragments are treated as OB, and thus do not have an endian. The figure below contains the sample pixel data of the table above in little endian format.



Further examples of encapsulated pixel data encoding are contained in Part 5 of the DICOM standard.

4.6.5. Icon Image Sequences

The Icon Image Sequence can contain small "thumbnail" images. This sequence also contains the Pixel Data Tag (7FE0,0010) just like the main message. Because this may or may not be compressed, some special considerations are necessary.

Sending on the network, writing a file, or writing a stream

1. Message is uncompressed, Icon is uncompressed.

   There is no user intervention required. An association will negotiate one of the unencapsulated transfer syntaxes, and both the image and the icon image will be sent as the negotiated unencapsulated transfer syntax.

2. Message is compressed, Icon is uncompressed.

   There is no user intervention required. An association will negotiate one of the encapsulated transfer syntaxes, and the image will be sent in this encapsulated transfer syntax, and the icon image will be sent `EXPLICIT_LITTLE_ENDIAN`. `EXPLICIT_LITTLE_ENDIAN` is the default syntax for the non-pixel data portion of a message (including nested pixel data) when the "main" pixel data is encapsulated.

3. Message is compressed, Icon is compressed.

   Minor intervention is required.

   Special creation of icon:

The only difference is that `MCitem.setTransferSyntax(ts)` must now be called upon creation of the `ICON_IMAGE` item so that you may register compression callbacks and utilize them.

Reading from network, a file, or a stream

No special conditions are required if the image is streamed in via `MCassociation.read(timeout)`, or `MCassociation.readToStream(timeout, io.IOBase)` methods. The sequence item automatically assumes:

`EXPLICIT_LITTLE_ENDIAN` if the pixel data contained in `ICON_IMAGE` is of defined length

-OR-

transfer syntax of the parent if the pixel data contained in `ICON_IMAGE` is of undefined length.

Duplicating messages

If duplicating from an unencapsulated to an encapsulated (compressed) transfer syntax, then the icon will be unencapsulated by default. To change the behavior, set `DUPLICATE_ENCAPSULATED_ICON` to Yes in the mergecom.pro file. This can be done dynamically (at run-time) via `MCconfig.setConfig()` method.

4.6.6. Validating Messages

Once your application has a populated message object, either one that you have built or one that you have received and are about to parse, Merge DICOM Toolkit supplies DICOM Toolkit DICOM message validation functionality. The `validate()` method will validate the specified message object instance against the DICOM Standard's specification for that service-command pair. If the method detects DICOM violations it returns a list of `MCvalidationError` objects describing the first violation.

Another file supplied with Merge DICOM Toolkit is the `message.txt` file. This file contains a listing of all the messages supported by the toolkit and the parameters they are validated against. `message.txt` is a useful guide in your application development because it specifies the attributes that can make up the object instance portion of each message type (service-command pair) and is often easier to use as a quick reference than paging through two or three parts of the DICOM Standard. `message.txt` also specifies the contents of items and files (see discussions of Sequence of Items and DICOM Files later in this document). Remember though that the DICOM Standard is the final word and that `message.txt` has its limitations as described further below.

`validate()` does not validate the attributes that make up the command portion of a DICOM message. Command attributes (attributes with a group number less than 0008) are also not specified in `message.txt`. The Merge DICOM Toolkit Library sets as many of the command group attributes as possible automatically. In some services, your application will need to set command attributes (e.g., the 'Affected SOP Class UID' attribute (0000,0002) in the C-MOVE response message). These special cases are described further in the Application Guides and in Part 7 of the DICOM Standard.

The toolkit validates the attribute set based on message definitions obtained from the DICOM database files (mrgcom3.dct and mrgcom3.dcm). This database is provided with the toolkit and it is updated with each version of the toolkit to reflect the latest changes and additions to the DICOM standard. Documentation on how the database can be amended is provided in the Message Database Manual (database.pdf).

Note that in case of some 1C or 2C attributes, the condition cannot be verified by the toolkit, in such cases the toolkit will consider the attribute of type 3

An excerpt of `message.txt` follows for the service-command pair `DETACHED_ PATIENT_ MANAGEMENT - N_ GET_ RSP` as an illustration. For each attribute in the message, at least one line of data is specified. This first line includes the tag, attribute name, value representation, and value type. Additional lines may be included for the attribute to list conditions, enumerated values, defined terms, and item names for attributes with a VR of SQ. You should refer to the DICOM Standard (parts 3 and 4) for a detailed description of particular conditions and their meanings.

```
################################################################
DETACHED_PATIENT_MANAGEMENT - N_GET_RSP
################################################################


0008,0005      Specific Character set                    CS 1C
Condition: EXPANDED_OR_REPLACEMENT_CHARACTER_SET_USED
Defined Terms:     ISO_IR 100, ISO_IR 101, ISO_IR 109, ISO_IR 110,
ISO_IR 144, ISO_IR 127, ISO_IR 126, ISO_IR 138, ISO_IR 148,
ISO_IR 166, ISO_IR 13, ISO 2022 IR 6, ISO 2022 IR 100,
ISO 2022 IR 101, ISO 2022 IR 109, ISO 2022 IR 110, ISO 2022 IR 144,
ISO 2022 IR 127, ISO 2022 IR 126, ISO 2022 IR 138, ISO 2022 IR 148,
ISO 2022 IR 149, ISO 2022 IR 166, ISO 2022 IR 13, ISO 2022 IR 87,
ISO 2022 IR 159, ISO_IR 192, GB18030
0008,1110              Referenced Study Sequence     SQ    2
Item Name(s): REF_STUDY
0008,1125              Referenced Visit Sequence     SQ    2
Item Name(s): REF_VISIT
0010,0010              Patient's Name                   PN 2
0010,0020              Patient ID                       LO 2
0010,0021              Issuer of Patient ID             LO 3
0010,0030              Patient's Birth Date             DA 2
0010,0032              Patient's Birth Time             TM 3
0010,0040              Patient's Sex                    CS 2
Enumerated Values: M, F, O
0010,0050              Patient's Insurance Plan Code SequenceSQ3
Item Name(s): CODE_SEQUENCE_MACRO
0010,1000              Other Patient IDs                LO 3
0010,1001              Other Patient Names              PN 3
0010,1005              Patient's Birth Name             PN 3
0010,1020              Patient's Size                   DS 3
0010,1040              Patient's Address                LO 3
0010,1060              Patient's Mother's Birth Name    PN 3
```

```
0010,1080               Military Rank                    LO 3
0010,1081               Branch of Service                LO 3
0010,1090               Medical Record Locator           LO 3
0010,2000               Medical Alerts                   LO 3
0010,2110               Allergies                        LO 3
0010,2150               Country of Residence             LO 3
0010,2152               Region of Residence              LO 3
0010,2154               Patient's Telephone Numbers      SH 3
0010,2160               Ethnic Group                     SH 3
0010,21A0               Smoking Status                   CS 3
Enumerated Values: YES, NO, UNKNOWN
0010,21B0               Additional Patient History       LT 3
0010,21C0               Pregnancy Status                 US 3
Enumerated Values: 0001, 0002, 0003, 0004
0010,21D0               Last Menstrual Date              DA 3
0010,21F0               Patient's Religious Preference   LO 3
0010,4000               Patient Comments                 LT 3
0038,0004               Referenced Patient Alias SequenceSQ 2
Item Name(s): REF_PATIENT_ALIAS
0038,0050               Special Needs                    LO 3
0038,0500               Patient State             LO       3
```

While Merge DICOM Toolkit's validation is not foolproof, it is very useful and will catch many standard violations. It validates the following:

- That the value assigned to an attribute is appropriate for that attributes VR.
- That all value type 1 attributes have a value, and that value is not null.
- That all value type 2 attributes have a value, and that value may be null.
- That a specified set of conditional attributes (value type 1C or 2C) are validated as value type 1 or 2 attributes when the specified condition is satisfied.
- That an attribute does not have too many or too few values for its specified value multiplicity.
- That an attribute that has enumerated values does not have a value that is not one of the enumerated values. A warning is also issued if an attribute that has defined terms has a value that is not one of those defined terms.
- That a non-private attribute is not included in the message that is not defined for that DICOM message (service-command pair).

As mentioned, Merge DICOM Toolkit does not capture all standard violations, and the DICOM Standard itself should be considered the final word when validating a message. Important limitations of Merge DICOM Toolkit validation include:

- DICOM Part 3 specifies Information Object Definitions (IODs) as being composed of modules. Each module contains attributes. Only in the case of composite IODs may an attribute be

specified in DICOM Part 3 as being contained in either a User Optional or Conditional Module. Merge DICOM Toolkit treats all such attributes as being value type 3 (optional).

- Also, only in the case of composite IODs (e.g., Ultrasound Image Object) used in storage services, may certain modules be mutually exclusive (e.g., curve and overlay modules). The attributes defined in these modules are all treated as type 3.
- For normalized services using the N-EVENT-REPORT command, the actual contents of an N-EVENT-REPORT message are dependent on the Event Type ID being communicated. Merge DICOM Toolkit treats all Event Type IDs identically when performing message validation; namely it treats all attributes as type 3.

Many times, validation is selectively used in an application as a runtime option or conditionally compiled into the source code. Validation might only be used during integration testing or in the field for diagnostic purposes. Reasons for this include performance since the overhead associated with message validation may be an issue, especially for larger messages having many attributes or on lower-end platforms. Also, validation can clutter the message log with warnings and errors that may not be desirable in a production environment. Performance issues related to message handling are discussed further under Message Exchange later in this document.

4.6.7. Streaming Messages

When DICOM messages are exchanged over a network, they are in an encoded format specified by the DICOM standard and the negotiated transfer syntax. Merge DICOM Toolkit calls this encoded format a **message stream** and supplies powerful functions that allow your applications to work directly with message streams.

When your application builds or parses messages as described earlier, it works with a Merge DICOM Toolkit message object. This message object abstracts and encapsulates the DICOM message and hides its details from the developer. When you send the DICOM message object over the network, Merge DICOM Toolkit internally creates a DICOM message stream that is passed over the network. This message stream is an encoded stream of bytes that follows all the rules of DICOM.

Merge DICOM Toolkit also supplies function calls to the developer to generate and read DICOM message streams directly. `MCdimseMessage.writeMessageToStream()` converts a message object to a message stream, while `MCdimseMessage.readMessageFromStream()` converts a message stream into a message object.

A `MCdimseMessage.writeMessageToStream()` method converts the attributes from (0008,0000) through (7FDF,FFFF) in the message object into a DICOM message stream using the explicit little endian transfer syntax. Explicit little endian transfer syntax is one of the three DICOM Transfer Syntaxes supported by Merge DICOM Toolkit. DICOM defines two other transfer syntaxes: implicit little endian (the default DICOM transfer syntax) and explicit big endian. See Part 5 of the DICOM Standard for a detailed description of transfer syntaxes.

Once your application has done the above and stored the stream somewhere, you could later rebuild a message object containing only group 0008 using `MCdimseMessage.readMessageFromStream()` method. This method converts only the attributes in group 0008 of the stream and places them in the. It is important that the transfer syntax specified in this call is identical to that used to create the stream or the call will fail with an error.

Message streams can be very valuable to your application for debugging and validation purposes. By writing DICOM message streams out to a binary file, you have a compact and reproducible representation of a message. You can directly examine the binary message stream to see how the data would be sent over the network. Also, you can read this binary file in again later to reconstruct

the original message object. Once you have the message object you can use the usual toolkit functions to examine or alter its contents.

Deflated Streams

The transfer syntax, Deflated Explicit VR Little Endian, gives you the ability to use the "deflate" algorithm to compress the entire data set. This transfer syntax was added mostly for structured reports, which are extremely redundant in their encoding, with considerable repetition of strings and tags. The toolkit uses zlib to implement deflate/inflate. This is an open source library that is built into the toolkit. Messages of this transfer syntax are still stored as message objects while the toolkit is handling them. Only when a message is "streamed" is the message deflated/inflated.

4.7. Message Exchange (Network Only)

4.7.1. General

We have discussed how associations are managed as well as how messages objects are populated and parsed. Now we'll discuss how these DICOM messages are exchanged with other application entities over the network.

The exchange of DICOM messages between AEs only occurs over an open association. After the DICOM client (SCU) application opens an association with a DICOM server (SCP), the client sends request messages to the server application. For each request message, the client receives back a corresponding response from the server. The server waits for a request message, performs the desired service, and sends back some form of status to the client in a response message.

The method for exchanging DICOM messages is:

`MCassociation.sendRequestMessage()` — The method sends a request message to the remote application. The message must have its service and command set, and the message command must be a valid DIMSE request command.

Some DICOM services require that values for certain command set attributes (i.e. group 0 attributes) be set. With the exceptions listed below, the application must set any command set attribute value before sending the message:

- The group length attribute (0000,0000) value is always set by the toolkit.
- If the message requires it, the Affected SOP Class UID attribute (0000,0002) value is set to the service's abstract syntax UID
- The command attribute (0000,0100) value is always set by the toolkit
- The Message ID attribute (0000,0110) value is set to a unique number for this association
- If the message requires it, the Priority attribute (0000,0700) value is set to "medium" priority (i.e. zero)
- The Data Set Type attribute (0000,0800) value is always set by the toolkit
- The Affected SOP Instance UID if required and not set by the application, is set by the toolkit based on the SOP Instance UID value in the data set

The `MCassociation.sendResponseMessage()` method may be called after receiving a request message.

The status parameter is used as the value for the Status (0000,0900) attribute and it provides the status of the requested operation. The value must be a valid response code for the service involved. Response codes are defined in the `MCresponse` class. Many DICOM services do not require a

response of more than just a status. This method can be used for services that require the setting of several message attributes (e.g. C_FIND_RSP).

`MCassociation.read()` — the method reads a DICOM DIMSE (DICOM Message Service Element) message from the network.

DICOM messages are exchanged over an active association connection. After successfully requesting an association or accepting an association request, a DICOM application must send or read DIMSE messages.

A DICOM service class user (SCU) sends request messages to the remote application to request that a service be performed. The service class provider (SCP) responds by sending one or more response messages back. The read method is used to retrieve these request and response messages.

When a message arrives, the toolkit returns an MCdimseMessage object representing both the DIMSE command set and the DIMSE data set which comprise the message. Methods of the returned MCdimseMessage provide access to the command set and data set attribute values of the message.

This method blocks until either a message is received or the number of seconds specified in the timeout parameter expires. If a communication error occurs, the association is released by the remote application or the association is aborted (on either side) this method throws an exception and logs an error message. Once the toolkit begins to receive a message, the INACTIVITY_TIMEOUT configuration value is used to determine if a transmission is stalled, meaning that the toolkit will abort the association if it does not receive more data from the network for the configured period of time. In that case this method will throw an exception.

`MCassociation.continueRead()` — the method partially reads the message from the network until the specified tag, starting from the previous ToTag call.

`MCassociation.readToStream()` and `MCassociation.continueReadToStream()` — the methods read a message to a io.IOBase data stream.

`MCassociation.sendSuccessResponse()` — sends a success response message to the remote application. This method may be called after successfully receiving a request message and can be used for DICOM services do not require a response of more than just a status. The value for the Status (0000,0900) attribute of the response message is set to 0 (success).

a. Sending Messages

The `MCassociation.sendResponseMessage()` includes parameter, `ResponseStatus`, that must be set to a valid DICOM response status. Example response status codes for the N_GET_RSP response message are summarized in the table below. Response codes for other DICOM commands are described in Part 4 of the DICOM Standard.

Table 4.4: Valid Response Message Status Codes for an N-GET Command

N_GET_RSP Status Codes
N_GET_SUCCESS
N_GET_WARNING_OPT_ATTRIB_UNSUPPORTED
N_GET_ATTRIBUTE_LIST_ERROR
N_GET_CLASS_INSTANCE_CONFLICT
N_GET_DUPLICATE_INVOCATION

N_GET_RSP Status Codes
N_GET_MISTYPED_ARGUMENT
N_GET_NO_SUCH_SOP_CLASS
N_GET_NO_SUCH_SOP_INSTANCE
N_GET_PROCESSING_FAILURE
N_GET_RESOURCE_LIMITATION
N_GET_UNRECOGNIZED_OPERATION

Your application may want to take advantage of Merge DICOM Toolkit's message validation functionality before sending a DICOM message out on the network, or before parsing and acting on a message received from some other device. Also, when constructing a request or response message, it is important to note that for some services, your application will need to set the value of command attributes in the message. Refer back to the Validating Messages section of this document for further discussion.

4.7.2. Asynchronous Communications

The DICOM standard defines an optional method for negotiation of an Asynchronous Operations Window. The Asynchronous Operations Window allows the client and server during association negotiation to define how many request messages can be sent over an association before a response message is required to be received. When the Asynchronous Operations Window is not negotiated (the default behavior of Merge DICOM Toolkit) only one request message can be sent before a response is received. Use of asynchronous operations can improve network performance when transferring a large number of messages over an association.

The specific fields negotiated over an association are the maximum number of operations, sub-operations, or notifications *invoked* by the requester of the association and the maximum number of operations, sub-operations, or notifications *performed* by the requester of the association. The client proposes settings for both of these fields, and the server responds with values less than or equal to the proposed values that are then used for the association.

The term *notifications* refers to N-EVENT-REPORT messages that are sent from an SCP to an SCU. These messages are used by DICOM services such as Print Job and Storage Commitment. The terms *operations* and *sub-operations* refer to all other message types. The term *sub-operations* specifically refers to services such as Query/Retrieve where multiple response messages are sent for a single request message.

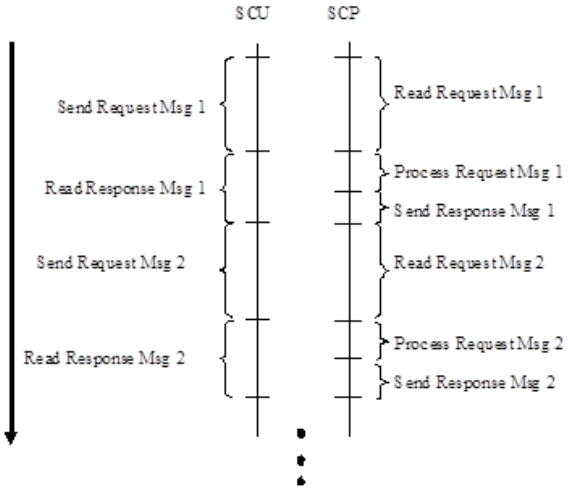
Asynchronous definitions

For a client negotiating the SCU role, the invoked field refers to the number of *operations* that could be sent without receiving a response message and the *performed* field would specify the maximum number of *notifications* received before the client is required to send a response message. For a client negotiating the SCP role and an asynchronous window, the invoked field would refer to the maximum *notifications* sent before receiving a response and the performed field would refer to the maximum *operations* received before the client is required to send a response.

Performance Tuning

Although asynchronous operations can be used for all DICOM service classes, this feature is most useful with the Storage Service Class. Asynchronous operations can be utilized to improve the network performance of transferring a large number of C STORE messages over an association. During normal synchronous operations, there typically is no network activity while a Storage SCU

waits for a response message from an SCP. Because there is a time when no data is being sent from the SCU to the SCP, a network is typically underutilized by synchronous DICOM transfers. Also, sending a large number of small images typically is slower than sending a smaller number of large images because a higher percentage of the association time is spent waiting on response messages. The figure below illustrates how an SCU waits on a response from the SCP while the SCP processes a message.

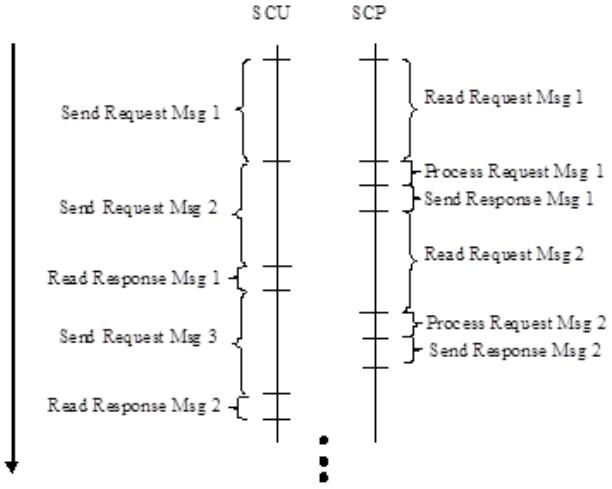


When asynchronous operations are negotiated, the SCU can poll for a response message, but if a message is unavailable, it can start sending the next request message right away (if the max operations will not be exceeded). This allows an SCU to utilize the network bandwidth more fully. There is only a small time when the SCU polls for a response message during which data is not being sent from the SCU to the SCP.

The majority of changes required to implement asynchronous communications are on the SCU side. A traditional SCP can effectively support asynchronous communications by simply enabling its negotiation over associations. It is possible, however, to do further optimization of SCP applications. On operating systems that Merge DICOM Toolkit supports threading, an SCP can be written to process messages in the background as it is reading request messages and to send response messages over the association when processing is completed. In this scenario, after reading a message, the SCP would pass the received message to another thread for processing and freeing. The main SCP thread would then go to reading the next message. Once processing is complete for a message, the background thread would signal the main thread to send a response message for the request. This allows the network bandwidth to be more fully utilized by having the SCP reading data off the network as much as possible. The SCP in this case must monitor the negotiated max operations so that it is not exceeded.

The figure below shows an example message exchange between an SCU and SCP when using asynchronous communications. This example shows how the SCU spends less time reading the

response messages and moves to sending the next request message to increase bandwidth utilization.



It is possible to create deadlock situations when writing an asynchronous application. A situation may arise where both the client and server are attempting to send data to each other that would cause a deadlock. In general, applications should poll for incoming messages before sending a new message. In particular, a Storage SCU should wait for a response message before sending a new request message. Because of the size of the messages exchanged, it is most likely that a Storage SCU would deadlock if it were not checking for response messages.

Performance Tuning

Finally, the configuration options `TCPIP_SEND_BUFFER_SIZE` and `TCPIP_RECEIVE_BUFFER_SIZE` are important for maximizing network performance. The send buffer specifies the amount of data that can be queued in the TCP/IP stack of the OS without actually being sent. The receive buffer specifies the amount of data that can be received by the TCP/IP stack of the OS before a Merge DICOM Toolkit application must start reading the data. These options allow data to be queued by the OS in the background while a Merge DICOM Toolkit application is doing other activities. For instance, an entire response message can usually be stored in the send buffer and a call to `MCassociation.sendResponseMessage()`. This allows the application to start preparing the next message to be sent while data is still being transferred. The maximum settings for these options is operating system dependent. It is suggested that these options be configured to the maximum setting for an operating system. If the settings are too high, an error will be logged to the merge.log file.

Use Full Duplex Networks with Asynchronous Communications

It is important that devices using asynchronous communications be configured to use full duplex network connections. Using asynchronous communications in half duplex mode would greatly degrade performance. Performance would more than likely be lower than if only synchronous communications were used.

4.8. Using Compression/Decompression

The purpose of registering a compression/decompression is to support compressed transfer syntaxes in DICOM more easily. Compressed transfer syntaxes are used to take advantage of the decreased image size that goes along with compressed pixel data. This is important when dealing with large images that need to be stored or transmitted across a network.

The compression/decompression, when registered, are utilized any time the functions are called, and the transfer syntax of a message has been set to JPEG_BASELINE, JPEG_EXTENDED_2_4, JPEG_LOSSLESS_HIER_14, JPEG_2000, JPEG_2000_LOSSLESS_ONLY or RLE.

If there is no compression/decompression registered, MCattributeSet attribute set getEncapsulatedFrame(), setEncapsulatedFrame() and getFrame() methods will work the same as getValue() and setValue(), except encapsulation delimiters will be removed and inserted, respectively.

MCattributeSet.setCompression(on) is used to activate the image compression/decompression functionality which takes in uncompressed pixel data and returns a compressed image, or takes in compressed data and returns the uncompressed pixel data.

Merge DICOM Toolkit has two sets of built in compressors and decompressors. The RLE compressor/decompressor can be used to compress/decompress data to/from an RLE transfer syntax encoding. These routines support data with photometric interpretation of MONOCHROME1, MONOCHROME2, RGB, and YBR_FULL.

The other Merge DICOM Toolkit built in set are the standard compressor and decompressor. These routines utilize libraries from Accusoft (formerly Pegasus Imaging Corporation) (**www.accusoft.com**). They support the JPEG_BASELINE, JPEG_EXTENDED_2_4, JPEG_LOSSLESS_HIER_14, JPEG_2000, JPEG_2000_LOSSLESS_ONLY transfer syntaxes with photometric interpretations MONOCHROME1, MONOCHROME2, RGB, and YBR. There are limits on the performance of the Pegasus libraries. These compressors are only available on platforms that Pegasus supports.

For JPEG_BASELINE, JPEG_EXTENDED_2_4, and JPEG_LOSSLESS_HIER_14, images can be compressed or decompressed at a maximum rate of three images (or frames) per second. For JPEG_2000 and JPEG_2000_LOSSLESS_ONLY, a dialog will be displayed on Windows each time the compressor or decompressor is used. For other platforms, a message will be displayed to stdout and a several second delay of several seconds will occur. Full licenses can be purchased from Accusoft and configured in Merge DICOM Toolkit to remove these compression and decompression limits. The licenses can be configured in the mergecom.pro configuration file.

The JPEG_BASELINE transfer syntax is UID 1.2.840.10008.1.2.4.50, JPEG Baseline (Process 1): Default Transfer Syntax for Lossy JPEG 8 Bit Image Compression, and uses Pegasus libraries 6420/6520. The table below details the photometric interpretation and bit depths supported by the standard compressor and decompressor for this transfer syntax. When lossy compressing RGB data, the standard compressor by default compresses the data into YBR_FULL_422 format. The compressor can also compress in YBR_FULL format if the COMPRESSION_RGB_TRANSFORM_FORMAT configuration option is set to YBR_FULL. The Photometric Interpretation tag must be changed by the application after compressing RGB data. Similarly, the Photometric Interpretation tag should be changed back to RGB before decompressing YBR_FULL or YBR_FULL_422 data.

Table 4.5: JPEG Baseline Supported Photometric Interpretations and Bit Depths

JPEG Baseline			
Photometric Interpretation	MONOCHROME1 MONOCHROME2	RGB	YBR_FULL_422
Bits Stored	8	8	8
Bits Allocated	8	8	8
Samples Per Pixel	1	3	3

NOTE: As of the present release of the toolkit, only the JPEG_BASELINE decompression is supported on the Android platform. The Pegasus library for JPEG_BASELINE compression (6420) is not available on the Android platform.

The JPEG_EXTENDED_2_4 transfer syntax is UID 1.2.840.10008.1.2.4.51, JPEG Extended (Process 2 & 4): Default Transfer Syntax for Lossy JPEG 12 Bit Image Compression (Process 4 only), and uses Pegasus libraries 6420/6520. The table below details the photometric interpretation and bit depths supported by the standard compressor and decompressor for this transfer syntax. When lossy compressing RGB data, the standard compressor by default compresses the data into YBR_FULL_422 format. The compressor can also compress in YBR_FULL format if the COMPRESSION_RGB_TRANSFORM_FORMAT configuration option is set to YBR_FULL. The Photometric Interpretation tag must be changed by the application after compressing RGB data. Similarly, the Photometric Interpretation tag should be changed back to RGB before decompressing YBR_FULL or YBR_FULL_422 data.

Table 4.6: JPEG Extended Supported Photometric Interpretations and Bit Depths

JPEG Extended (Process 2 & 4)					
Photometric Interpretation	MONOCHROME1 MONOCHROME2			RGB	YBR_FULL_422
Bits Stored	8	10	12	8	8
Bits Allocated	8	16	16	8	8
Samples Per Pixel	1	1	1	3	3

NOTE: As of the present release of the toolkit, only the JPEG_EXTENDED_2_4 decompression is supported on the Android platform. The Pegasus library for JPEG_EXTENDED_2_4 compression (6420) is not available on the Android platform.

The JPEG_LOSSLESS_HIER_14 transfer syntax is UID 1.2.840.10008.1.2.4.70, JPEG Lossless, Non-Hierarchical, First-Order Prediction (Process 14 [Selection Value 1]): Default Transfer Syntax for Lossless JPEG Image Compression, and uses Pegasus libraries 6220/6320. The table below details the photometric interpretation and bit depths supported by the standard compressor and decompressor for this transfer syntax. The standard compressor does not do a color transformation to RGB data when compressing with JPEG_LOSSLESS_HIER_14. The Photometric Interpretation tag should be left as RGB in this case.

Table 4.7: JPEG Lossless Supported Photometric Interpretations and Bit Depths

JPEG Lossless Non-Hierarchical Process 14			
Photometric Interpretation	MONOCHROME1 MONOCHROME2	RGB YBR_FULL	PALETTE COLOR
Bits Stored	2 to 16	8	1 - 16
Bits Allocated	8 or 16	8	8 or 16
Samples Per Pixel	1	3	1

NOTE: As of the present release of the toolkit, only the `JPEG_LOSSLESS_HIER_14` decompression is supported on the Android platform. The Pegasus library for `JPEG_LOSSLESS_HIER_14` compression (6220) is not available on the Android platform.

The `JPEG_2000` transfer syntax is UID 1.2.840.10008.1.2.4.91, JPEG 2000 Image Compression, and uses Pegasus libraries 6820/6920 for lossy or lossless. The table below details the photometric interpretation and bit depths supported by the standard compressor and decompressor for this transfer syntax.

Table 4.8: JPEG 2000 Lossy Supported Photometric Interpretations and Bit Depths

JPEG 2000 (When used for Lossy)							
Photometric Interpretation	MONOCHROME1 MONOCHROME2				YBR_ICT	RGB	YBR_FULL
Bits Stored	8	10	12	16	8	8	8
Bits Allocated	8	16	16	16	8	8	8
Samples Per Pixel	1	1	1	1	3	3	3

NOTE: As of the present release of the toolkit, only the `JPEG_2000` decompression is supported on the Android platform. The Pegasus library for `JPEG_2000` compression (6820) is not available on the Android platform.

The `JPEG_2000_LOSSLESS_ONLY` transfer syntax is UID 1.2.840.10008.1.2.4.90, JPEG 2000 Image Compression (Lossless Only), and uses Pegasus libraries 6820/6920 for lossless. The table below details the photometric interpretation and bit depths supported by the standard compressor and decompressor for this transfer syntax.

Table 4.9: JPEG 2000 Lossless Supported Photometric Interpretations and Bit Depths

JPEG 2000 Lossless							
Photometric Interpretation	MONOCHROME1 MONOCHROME2				YBR_RCT YBR_FULL	RGB	PALETTE COLOR
Bits Stored	8	10	12	16	8	8	1 - 16
Bits Allocated	8	16	16	16	8	8	8 or 16
Samples Per Pixel	1	1	1	1	3	3	1

NOTE: As of the present release of the toolkit, only the `JPEG_2000_LOSSLESS_ONLY` decompression is supported on the Android platform. The Pegasus library for `JPEG_2000_LOSSLESS_ONLY` compression (6820) is not available on the Android platform.

NOTE: When using the standard compressor, all data needs to be right justified, i.e. bit 0 contains data, but the highest bits may not. RGB and YBR must be non-planar (R1G1B1, R2G2B2, ... or Y1Y2B1R1, Y3Y4B3R3...)

NOTE: `JPEG_2000`/`JPEG_2000_LOSSLESS_ONLY` will cause an irreversible, or reversible color transformation when compressing RGB data. The Photometric Interpretation MUST be changed from RGB to:

- `YBR_ICT` if `JPEG_2000` is used with `COMPRESSION_WHEN_J2K_USE_LOSSY = Yes` (Lossy color transform for lossy compression)
- `YBR_RCT` if `JPEG_2000_LOSSLESS_ONLY` or `JPEG_2000` are used with `COMPRESSION_WHEN_J2K_USE_LOSSY = No` (Lossless color transform for lossless compression).

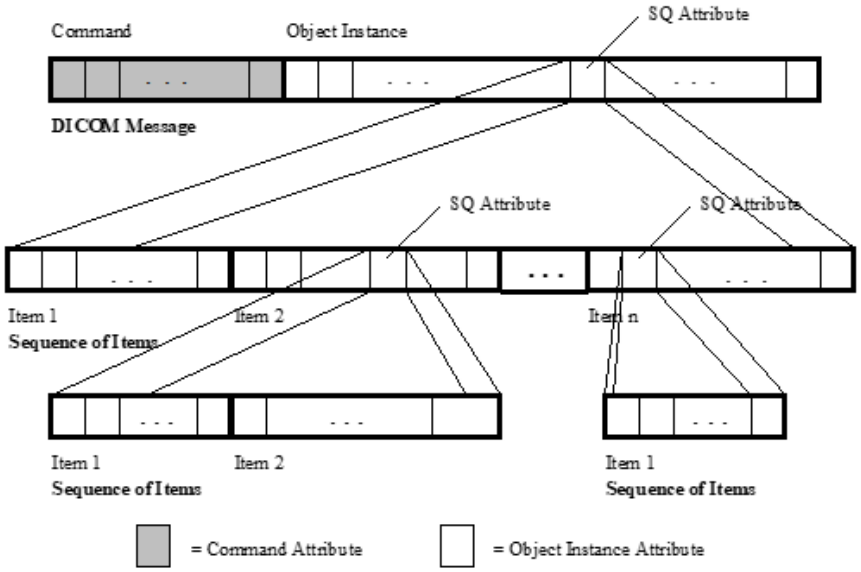
Similarly, on the decompression end, the Photometric Interpretation should be changed back to `RGB`, but the Lossy Image Compression attribute should indicate it has been lossy compressed.

The `UPDATE_GROUP_0028_ON_DUPLICATE` configuration option can also be enabled so Merge DICOM Toolkit will update the Group 0x0028 tags for you. When this configuration option is enabled, the Photometric Interpretation will be updated for you as mentioned above. When decompressing an image, the photometric interpretation will also be updated. In addition, when lossy compression is done, the Lossy Image Compression, Lossy Image Compression Ratio, and Lossy Image Compression Method tags will be updated by Merge DICOM Toolkit.

4.9. Sequences of Items

The DICOM Value Representation SQ is used to indicate an attribute in a DICOM message containing a value that is a sequence of items. A sequence of items is a set of object instances, where each object instance can also contain attributes that have a VR of SQ. This powerful capability allows the nesting of objects, or the definition of 'container' objects (such as folders, film boxes, directories, etc.). One can think of these nested objects as message objects minus the command portion.

The figure below shows a DICOM message containing a sequence of items running two levels deep. Note that these nested sequences are contained within the same Message Stream. Sequences of items can also be contained in a DICOM file, and we will see that they are contained in DICOMDIR files. An attribute whose value is a sequence of items is simply an attribute that has a potentially large and complex value. Fortunately, Merge DICOM Toolkit allows your application to deal with sequences of items an item at a time and hierarchically, as pictured in the figure below, and takes care of the encoding of the sequence within the DICOM message stream.



Encoding and decoding attributes in an item

Each item object in a sequence is a special **MCitem** class derived from MCattributeSet class. All the message building and parsing functionality described in previous sections of this manual also applies to item objects. The MCattributeSet's **getValue** and **setValue** methods work on item (MCitem) objects as well as message (MCdimseMessage) objects.

`MCitem` object contains strings used to identify each item and are listed in the `message.txt` file for attributes having a VR of SQ. The contents of each item are also listed in the `message.txt` file. Below are two excerpts of `message.txt`, one showing a reference to the Issuer of Accession Number Item, and the other the contents of that item.

```
#####################################################################
CHEST_CAD_SR - C_STORE_RQ
#####################################################################

0008,0005    Specific Character Set                        CS    1C
Condition: EXTENDED_OR_REPLACEMENT_CHARACTER_SET_USED
Defined Terms:     ISO_IR 100, ISO_IR 101, ISO_IR 109, ISO_IR 110,
ISO_IR 144, ISO_IR 127, ISO_IR 126, ISO_IR 138, ISO_IR 148,
ISO_IR 166, ISO_IR 13, ISO 2022 IR 6, ISO 2022 IR 100,
ISO 2022 IR 101, ISO 2022 IR 109, ISO 2022 IR 110, ISO 2022 IR 144,
ISO 2022 IR 127, ISO 2022 IR 126, ISO 2022 IR 138, ISO 2022 IR 148,
ISO 2022 IR 149, ISO 2022 IR 166, ISO 2022 IR 13, ISO 2022 IR 87,
ISO 2022 IR 159, ISO_IR 192, GB18030
0008,0012    Instance Creation Date                        DA    3
0008,0013    Instance Creation Time                        TM    3
0008,0014    Instance Creator UID                          UI    3
0008,0015    Instance Coercion DateTime                    DT    3
0008,0016    SOP Class UID                                 UI    1
0008,0018    SOP Instance UID                              UI    1
0008,001A    Related General SOP Class UID                 UI    3
0008,001B    Original Specialized SOP Class UID            UI    3
0008,0020    Study Date                                    DA    2
0008,0021    Series Date                                   DA    3
0008,0023    Content Date                                  DA    1
0008,0030    Study Time                                    TM    2
0008,0031    Series Time                                   TM    3
0008,0033    Content Time                                  TM    1
0008,0050    Accession Number                              SH    2
0008,0051    Issuer of Accession Number Sequence           SQ    3
```

```
Item Name(s): ISSUER_OF_ACCESSION_NUMBER
...
...

================================================================
Item Name: ISSUER_OF_ACCESSION_NUMBER
================================================================

0040,0031    Local Namespace Entity ID                    UT    1C
Condition: A00400032_NOT_PRESENT
0040,0032    Universal Entity ID                          UT    1C
Condition: A00400031_NOT_PRESENT
0040,0033    Universal Entity ID Type                     CS    1C
Condition: A00400032_PRESENT
Defined Terms:     DNS,EUI64,ISO,URI,UUID,X400,X500
```

Encoding items in a sequence

To encode a sequence item into an attribute of Value Representation SQ, treat the attribute as a multi-valued attribute, where each value is an MCitem object. The following sample code fragment gives an example of encoding an Icon Image Item into a sequence:

```
item = MCitem()
item.setName(MCitems.ICON_IMAGE)

item.setValue(MCdicom.SPECIFIC_CHARACTER_SET, 'ISO 2022 IR 13')
item.addValue(MCdicom.SPECIFIC_CHARACTER_SET, 'ISO 2022 IR 87')
item.setValue(MCdicom.REFERRING_PHYSICIANS_NAME, 'REF PHYSICIAN')

msg.addValue(MCdicom.REFERENCED_IMAGE_SEQUENCE, item)
```

4.10. DICOM Files

Maintaining a DICOM file set is a matter of maintaining various DICOM files and a single DICOM directory file (DICOMDIR). First, the functions supplied by Merge DICOM Toolkit that operate on all DICOM files are described; followed by a description of those functions that are especially suited for the complexities of the DICOMDIR file.

4.10.1. File System Interface Functions

This may sound strange, but all the media interchange functionality of the DICOM Toolkit relies on functions that you supply to interface with the particular physical medium and file system format on

your target device. This approach was chosen because of the wide variety of media and file system configurations allowed by the DICOM Standard and the potentially unlimited combination of media devices, device drivers, and file system combinations for which DICOM media interchange applications may be developed.

DICOM Toolkit provides powerful DICOM media functionality by supplying your application with:

- A greatly simplified way to deal with the complex encoding and decoding required within a DICOM file.
- Methods that are very consistent with that used for the maintenance of DICOM messages used in network functionality; many of the encoding and decoding functions already described apply equally well to DICOM file objects.

To perform all this functionality on your medium of choice, you need only supply the two file system interface functions just discussed.

4.10.2. Creating a File Object

Before the contents of an existing DICOM file can be read in or a new DICOM file can be created, an instance of `MCfile` derived from `McattributeSet` class must be created. Its constructor `MCfile()` creates the file object whose type is specified by the supplied service-command pair with `MCfile.setServiceCommand(service, command)` method. For example, the following call creates an DICOM CT image file object:

```
file = MCfile()


file.setName('DICOM_FILE')
file.setServiceCommand(MCservices.STANDARD_CT, MCcommand.C_STORE_RQ)


file.setValue(MCdicom.SPECIFIC_CHARACTER_SET, 'ISO 2022 IR 13')
file.addValue(MCdicom.SPECIFIC_CHARACTER_SET, 'ISO 2022 IR 87')
```

STANDARD_CT is the service name, and C_STORE_RQ is the command name identifying the class of file object being created (see **TABLE 2.5: SERVICE-COMMAND PAIRS SPECIFYING OBJECT INSTANCES THAT CAN BE STORED IN A DICOM FILE ON PAGE 28**).

It is important to realize that `MCfile()` does not create the physical DICOM file out on the medium; the `MCfile.writeP10File()` method does that. `MCfile()` corresponds to the `MCdimseMessage()` used in networking; it creates references to the proper message info file along with the data dictionary and builds an unpopulated file object instance for your application to fill in. This file object contains empty attributes.

When your application is done using a file object, the file object should be freed using the `MCfile.dispose()` method.

4.10.3. Reading Files

To read in the contents of a DICOM file for analysis or parsing, you must open the file. Opening a DICOM file in the DICOM Toolkit API means that a complete file object is filled in from an existing physical file. This means the entire DICOM file is read in on the open.

The following code reads a file into the file object:

```
file = MCfile()

file.readP10File('file.dcm')
```

Once the file object has been read, the file's `MCfile.getValue()` and `MCfile.setValue()` methods can be used to read and set attribute values from the file object.

Performance Tuning

Variants of `MCfile readP10File()` are supported by the toolkit:

- `readP10FileBypassBulk()` will read in all attribute's value but will not store the data of type OB or OW. This method can be used to increase performance for handling attributes of type OB, OW, OL, OV, OD or OF. If the application associated with this file object (see `setApplication()` method) has a value storage provider registered for an attribute of type OB, OW, OL, OV, OD or OF, this method will pass the offset of the attribute's value from the beginning of the file along with length of the value to the value storage through the `MCIvalueStorage.receiveDataLength()` method. When the data is needed by the user or the toolkit, the value storage can retrieve it from the media.
- If the `EndTag` parameter is specified `readP10File()` will stop reading the attributes from media into the file object when it reaches the first attribute greater than a specified tag. The offset in bytes from the beginning of the file to the beginning of the first attribute greater than the specified tag is returned. The user's application must then deal with reading in the rest of the DICOM file from media. This function is most useful when the DICOM file contains pixel data (7FE0, 0010) as its last attribute and this pixel data is very large. In these instances, you may wish to ignore the pixel data, read it in later using callback mechanism, or process it directly from the file using your own special filters or hardware.
- If `Bypass` parameter is `True`, `readP10File()` will read the attributes from the media into the file object including a specified tag, but will not read the tag attributes value. This function is most useful when the DICOM file contains large pixel data (7FE0, 0010) attribute. The user's application callback must then deal with reading data from seekable DICOM file stream from the given offset provided in the REQUEST_FOR_DATA_WITH_OFFSET command.

Once a DICOM `MCfile` file object has been created and read, the file can be written out to media using a single `MCfile.writeP10File` method.

4.10.4. File Validation

File validation occurs in much the same manner as message validation. Before the file can be validated it must be read into a file object and `MCfile.setServiceCommand(service, command)` method must be called to identify the type of file object before validation can occur. Please see the earlier discussion of message validation, as almost all of it applies equally well to file validation.

Performance Tuning

DICOM file validation does involve processing overhead. The most significant overhead is in the accessing of the message info files, and significantly less overhead is involved in actually validating the contents of the file object structure. It is **important** to understand that depending on the way in which your message object was created, this validation overhead can occur at different points in your application.

Many times, `MCfile.validate()` is selectively used in an application. Validation might only be used during integration testing or in the field for diagnostic purposes. Reasons for this include performance since the overhead associated with file validation may be an issue, especially for larger files having many attributes or on lower-end platforms. Also, validation can clutter the message log with warnings and errors that may not be desirable in a production environment.

4.10.5. Converting Files to/from Messages

Two very useful and powerful functions are supplied for converting file objects to message objects and vice versa. These methods are `MCfile.fromMessage()` and `MCdimseMessage.fromFile()`. Remember that most DICOM files (other than the DICOMDIR file) are simply the information object portion of a DICOM message encapsulated within a DICOM file (surrounded by a file preamble, meta information, and optional padding).

These two functions are most often useful when reading and writing image files to and from DICOM media that were received (or will need to be transmitted) over the network as C-STORE request messages.

4.11. Private Attributes

Private attributes supply a mechanism for applications to extend standard message objects and were discussed earlier in this document. Private attributes in message objects are handled in much the same way as standard attributes with three major exceptions:

- Standard attributes in the Merge DICOM Toolkit API are referenced by Tag, while private attributes are referred to by `PrivateCode`, `Group`, and `ElementByte`.
- The `Group` number of a private attribute must always be odd and is the 4 high order hexadecimal digits of the private Attribute Tag, while for a standard attribute it is always even.
- Private Code is the Private Creator code that will be used to identify the block of 0xFF (255) private attributes you are reserving within private Group. Private Code can be composed of up to 64 alphanumeric characters and the space character.
- Element Byte is the 2 hexadecimal digits identifying the private attribute within the Private Block identified by Private Code.

Up to 255 other private attributes could be added to the ACME_IMG_CORP private block in group 0x1455 using the above call and `ElementByte` values of 01 through FF. If more attributes are required, another private block (with a different `PrivateCode`) will need to be added.

`PrivateCodes` must be used to refer to private attributes, because private blocks may be placed in different locations within a private group, depending on what other blocks of private attributes have already been reserved. `PrivateCodes` are a way to refer to these blocks, independently of their physical location in the message stream.

Adding private attributes to a message

Once private attributes have been added with `MCattributeSet.addAttribute()` method, their values can be assigned and retrieved identically to standard attributes using `MCattributeSet.getValue()` and `MCattributeSet.setValue()` methods. The next example shows how to create and add a private attribute tag:

```
tag = MCtag(0x155590001, 'ACME_IMG_CORP')

msg.addAttribute(tag, MCvr.LO)

msg.setValue(tag, 'PRIVATE VALUE 1')

msg.addValue(tag, 'PRIVATE VALUE 2')
```

4.12. Multi-threading Support

The Merge DICOM Toolkit library has been designed to be thread safe.

There are some assumptions, however, concerning the thread safety of Merge DICOM Toolkit. In most cases, it is assumed that a Merge DICOM Toolkit object is only accessed from one thread at a time. This applies to Message objects, file objects, item objects, and association objects. Note, however, that different instances of an object can always be manipulated in different threads at the same time.

There are exceptions to this rule, however. The following is a summary of them:

The `sendRequest()`, `sendResponse()`, `read()`, `readToStream()`, `release()` and `abort()` methods of the `MCassociation` class can all be used in one thread while another thread is calling `MCassociation.abort()` within another thread. This is useful for allowing a user to asynchronously cancel an in-progress association.

It is also possible to access tags within a message as it is being read from the network. It is possible to call the `MCattributeSet.getValue()` and `MCattributeSet.setValue()` methods from one thread while another thread is calling `MCassociation read()`, `continueRead()` and `continueReadToStream()` methods.

4.13. Memory Management

Performance Tuning

Merge DICOM Toolkit contains its own memory management routines that are optimized for how it uses memory. They have been adapted to manage specific data structures that are frequently allocated by the toolkit. These include but are not limited to data structures for associations, messages, and tags. The memory management routines have the characteristic that they do not actually "free" the memory that has been acquired. Instead, they mark the data as being free and place the memory in a list for reuse later. These routines have been optimized to quickly acquire and free memory being used by the toolkit. They also allow Merge DICOM Toolkit to not depend on the memory management of a particular operating system.

These memory routines have also been extended for use with variable sized memory buffers. Merge DICOM Toolkit uses these routines to allocate buffers in sizes between 4 bytes and 28K. When an allocation is requested, the toolkit will take the smallest buffer that will fit the bytes requested. These buffers will be kept in the toolkit's internal memory pool and never freed. For allocations larger than 28K, Merge DICOM Toolkit will simply use the 'C' functions `malloc()` and `free()`. Under most conditions, Merge DICOM Toolkit breaks up large DICOM data elements such as pixel data into chunks of data smaller than 28K so that they can be managed through these routines.

The end result of these routines is that applications using Merge DICOM Toolkit typically expand to the maximum amount of memory used at one time. The total memory allocation will not shrink from this point. In applications that repeatedly perform a consistent operation, the memory being used by Merge DICOM Toolkit should stabilize and not increase in size. In applications using Merge DICOM Toolkit from multiple threads, this memory usage is not as consistent and depends on the timing of the threads using the toolkit. As a result of these routines, the first time an application performs a DICOM operation is typically slower than subsequent operations.

Merge Python Toolkit supplies the `MC.reportMemory()` and `MC.cleanMemory()` methods to allow some user control over this memory management. `MC.reportMemory()` reports how much memory is currently allocated and in use by the toolkit. The `MC.cleanMemory()` routine can be used to actually free memory allocated by Merge DICOM Toolkit. The routine looks for blocks of memory that are no longer in use by the toolkit and frees them with the operating system. This can be useful when a Merge DICOM Toolkit application reads a large DICOM object (such as a large DICOMDIR or a large multi-frame image) and the user would like to free some of the memory associated with the object.

When developing a DICOM application with Merge DICOM Toolkit, the most memory intensive operation is dealing with image data. The following sections discuss various Merge DICOM Toolkit functions. A description is given of how these functions manage memory in conjunction with various toolkit configuration settings.

4.13.1. Assigning Pixel Data

`MCattributeSet.setValue()` method is used to assign OB, OW, OL, OV, OF or OD data to a DICOM tag. These value representations are used to store image data or other large data elements.

Data can be passed to Merge DICOM Toolkit in several ways. The entire data value can be passed in a single `MCattributeSet.setValue()` call, or the data can be supplied in chunks using value storage mechanisms (see `MCIstorageProvider` and `MCIvalueStorage` classes). When passed data, the toolkit will allocate a buffer the size of the chunk received and copy the data into this buffer for storage. If the data is passed in chunks smaller than 28K, Merge DICOM Toolkit's internal memory management code will be used. If the chunks are larger than 28K, `malloc()` will be used to allocate the storage for the buffers. If large images are being handled, it may be desirable to pass these data in chunks larger than 28K, so the memory is freed after processing of the image has been completed. This will keep the nominal memory usage of Merge DICOM Toolkit lower. When providing data in chunks smaller than 28K, it is recommended that sizes of 16K, 20K, 24K, or 28K be used. Using these size chunks will reduce the overhead in storing the data.

The `MCattributeSet.setValue()` method can also be directed to store data in temporary files. The `LARGE_DATA_STORE` and `LARGE_DATA_SIZE` configuration options in the mergecom.pro file dictate when data is stored in temporary files. When the `LARGE_DATA_STORE` option is set to `FILE`, data elements that are larger than configured by the `LARGE_DATA_SIZE` option are stored in temporary files. The size of the buffer passed to value storage does not have an effect on memory usage.

4.13.2. Reading Messages from the Network

Merge DICOM Toolkit has a single function for reading messages from the network. `MCassociation.read()` creates a message object and loads the message into memory while reading from the network. When using Merge DICOM Toolkit's standard memory management routines, the method for storing the image data can be influenced.

Data are read from the network by PDUs. However, they are stored internally in sizes dictated by the `WORK_BUFFER_SIZE` configuration value. If a chunk of data read is smaller than the value for the `WORK_BUFFER_SIZE`, the chunk will simply be stored. If it is larger, the data will be stored internally in `WORK_BUFFER_SIZE` buffers.

By supporting a maximum PDU size and `WORK_BUFFER_SIZE` larger than 28K, Merge DICOM Toolkit will store the buffers in memory allocated with the native C function `malloc()`. This can be used to reduce the toolkit's typical memory usage. Note, however, that SCU systems do not necessarily size their PDUs according to the Maximum PDU size negotiated. This solution does not guarantee that image data will be stored with `malloc()`.

4.13.3. Loading Messages from Disk

This functionality shares the same characteristics as when data are being read from the network with `MCassociation.read()`.`MCassociation.readToStream()` is used to read DICOM "stream" objects and `MCfile.readP10File()`,`MCfile.readP10FileBypassBulk()` are used to read DICOM Part 10 format files. Whereas objects being read from the network determine

memory usage by the PDU size, these functions determine memory usage by the size of the buffers passed from their callback functions. The `WORK_BUFFER_SIZE` configuration value has the same impact as when reading from the network.

If the data are stored in file format, the `MCfile.readP10File()` and `MCfile.readP10FileBypassBulk()` functions can be used to leave the image data on disk until they are sent over the network.

a. Saving Received Images Directly to Disk

In conjunction with the registered callback function, data can also be stored directly to disk when it is being read. The image header data can be written to disk from within the registered callback. The user must write the attribute tag, value representation if needed, and the length of the image data attribute to the file. The image data is written to the file in subsequent calls to the user's registered callback function.

When `MCassociation.read()` is parsing a message being received, it will notify the user's registered callback function when it has parsed the header information and determines the image data's length. The registered callback function will be called with the `PROVIDING_DATA_LENGTH` flag and is supplied the Message ID of the message being read. At this point, the user can stream the header file to disk with `MCassociation.readToStream()`. As the image data is received, it can be added to the end of this file.

Data can also be stored as DICOM files with this method. The message cannot be converted into a file object at this point with `Mcfile.fromMessage()` as would normally be done. So, a separate file must be created to add the DICOM Part 10 Meta Header information. This header can be written out from within the callback. After the end of the meta header, the message can be streamed to disk with a call to `MCassociation.readToStream()` in the transfer syntax specified in the Meta Header. As subsequent image data is passed to the user's callback function, the data can be written to file. Because the endian of the transfer syntax being written may be different than the endian of the system being used, there may be a need for byte swapping of the pixel data in this implementation.

There is a potential risk with this implementation. Although the current definition of the DICOM image types does not include any data elements after the pixel data, future versions may add data elements there.

4.14. DICOM Structured Reporting

The Merge DICOM Toolkit provides high-level functionality to handle DICOM Structured Report (SR) Documents. This functionality provides a simple way for encoding and decoding SR Document content by manipulating content items and their attributes instead of tags and values.

4.14.1. Structured Report Structure and Modules

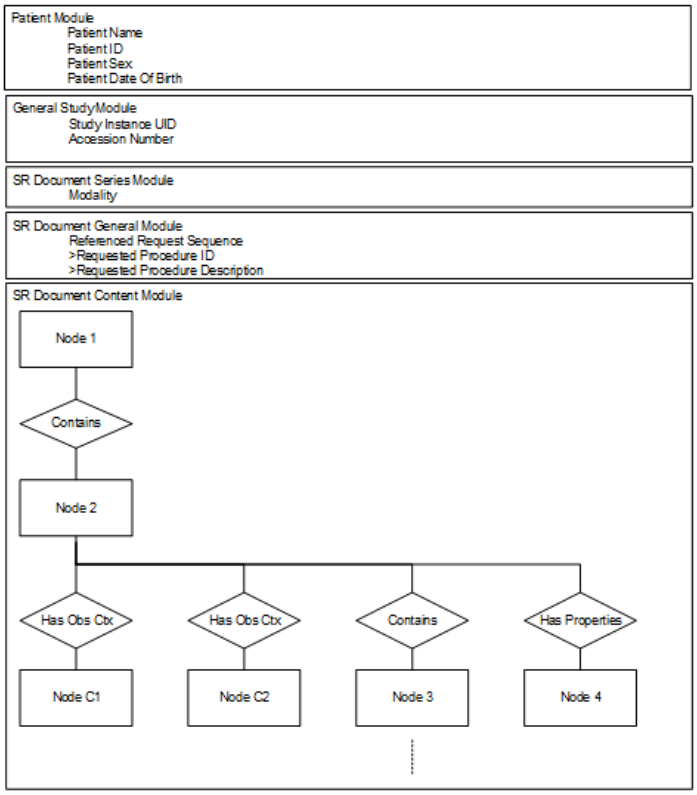
The DICOM standard Part 3 defines the following generic types of SR Information Object Definitions (IODs):

- Basic Text SR Information Object Definition — The Basic Text Structured Report (SR) IOD is intended for the representation of reports with minimal usage of coded entries (typically used in Document Title and headings) and a hierarchical tree of headings under which may appear text and subheadings. Reference to SOP Instances (e.g. images or waveforms or other SR Documents) is restricted to appear at the level of the leaves of this primarily textual tree. This

structure simplifies the encoding of conventional textual reports as SR Documents, as well as their rendering.

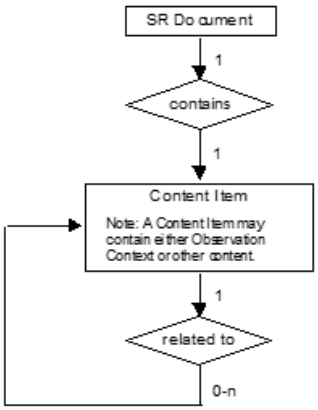
- Enhanced SR Information Object Definition — The Enhanced Structured Report (SR) IOD is a superset of the Basic Text SR IOD. It is also intended for the representation of reports with minimal usage of coded entries (typically Document Title and headings) and a hierarchical tree of headings under which may appear text and subheadings. In addition, it supports the use of numeric measurements with coded measurement names and units. Reference to SOP Instances (e.g. images or waveforms or SR Documents) is restricted to appear at the level of the leaves of this primarily textual tree. It enhances references to SOP Instances with spatial regions of interest (points, lines, circle, ellipse, etc.) and temporal regions of interest.
- Comprehensive SR Information Object Definition — The Comprehensive SR IOD is a superset of the Basic Text SR IOD and the Enhanced SR IOD, which specifies a class of documents, the content of which may include textual and a variety of coded information, numeric measurement values, references to the SOP Instances and spatial or temporal regions of interest within such SOP Instances. Relationships by-reference are enabled between Content Items.

There are more specific SR IODs defined in the DICOM, like *Key Object Selection Document* and *Mammography CAD SR*. Those IODs use the same way to encode data and the difference is in the constrains on the Content Item Types and their relationships. The figure below illustrates the typical SR Document structure. As you can see, the top level header is very similar to the DICOM image IODs and consists of the same Patient, Study and Series modules. The main difference from other IODs is the *SR Document Content Module*. The attributes in this Module convey the content of an SR Document.



a. The SR Document Hierarchy

The Document Content Module has a tree structure and consists of a single root Content Item (Node 1) that is the root of the SR Document tree. The root Content Item conveys either directly or indirectly all of the other nested Content Items in the document. The hierarchical structuring of the Content Tree provided by recursively nesting Content Items. A parent (or source) Content Item has an explicit relationship to each child (or target) Content Item, conveyed by the Relationship Type. The figure below depicts the relationship of SR Documents to Content Items and the relationships of Content Items to other Content Items and to Observation Context.



Each Content Item contains the following:

- A name/value pair, consisting of:
  - a single Concept Name Code that is the name of a name/value pair or a heading; and
  - a value (text, numeric, code, etc.).
- References to images, waveforms or other composite objects, with or without coordinates.
- Relationships to other Items, either by-value through nested Content Sequences, or by-reference.

NOTE: Some Content Item Types can have multiple values.

4.14.2. Content Item Types

The table below defines all possible Content Item Types that can be used in the SR Document Content Module. The choice of which may be constrained by the IOD in which this Module is

contained. Merge DICOM Toolkit Definition column specifies the enumerated value used in the Toolkit to identify the Content Item Type.

Table 4.10: SR Content Item Types

Item Type	Merge DICOM Toolkit Class	Concept Name	Description
TEXT	MCtextItem	Type of text, for example, "Findings", or name of identifier, for example, "Lesion ID"	Free text, narrative description of unlimited length. May also be used to provide a label or identifier value.
NUM	MCnumItem	Type of numeric value or measurement, for example, "BPD"	Numeric value fully qualified by coded representation of the measurement name and unit of measurement.
CODE	MCcodeItem	Type of code, for example, "Findings"	Categorical coded value. Representation of nominal or non-numeric ordinal values.
DATETIME	MCdateTimeItem	Type of DateTime, for example, "Date/Time of onset"	Date and time of occurrence of the type of event denoted by the Concept Name.
DATE	MCdateItem	Type of Date, for example, "Birth Date"	Date of occurrence of the type of event denoted by the Concept Name.
TIME	MCtimeItem	Type of Time, for example, "Start Time"	Time of occurrence of the type of event denoted by the Concept Name.
UIDREF	MCuidReferenceItem	Type of UID, for example, "Study Instance UID"	Unique Identifier (UID) of the entity identified by the Concept Name.
PNAME	MCpersonNameItem	Role of person, for example, "Recording Observer"	Person name of the person whose role is described by the Concept Name.
COMPOSITE	MCcompositeItem	Purpose of Reference	A reference to one Composite SOP Instance which is not an Image or Waveform.
IMAGE	MCimageItem	Purpose of Reference	A reference to one Image. IMAGE Content Item may convey a reference to a Softcopy Presentation State associated with the Image.

Item Type	Merge DICOM Toolkit Class	Concept Name	Description
WAVEFORM	MCwaveformItem	Purpose of Reference	A reference to one Waveform.
SCOORD	MCspatialCoordinatesItem	Purpose of Reference	Spatial coordinates of a geometric region of interest in the DICOM image coordinate system. The IMAGE Content Item from which spatial coordinates are selected is denoted by a SELECTED FROM relationship.
SCOORD3D	MCspatialCoordinates3DItem	Purpose of Reference	3D spatial coordinates (x, y, z) of a geometric region of interest in a Reference Coordinate System.
TCOORD	MCtemporalCoordDateTimeItem MCtemporalCoordPositionsItem MCtemporalCoordTimeOffsetsItem	Purpose of Reference	Temporal Coordinates (i.e. time or event based coordinates) of a region of interest in the DICOM waveform coordinate system. The WAVEFORM or IMAGE or SCOORD Content Item from which Temporal Coordinates are selected is denoted by a SELECTED FROM relationship.
CONTAINER	MCcontainerItem	Document Title or document section heading. Concept Name conveys the Document Title (if the CONTAINER is the Document Root Content Item) or the category of observation.	CONTAINER groups Content Items and defines the heading or category of observation that applies to that content. The heading describes the content of the CONTAINER Content Item and may map to a document section heading in a printed or displayed document.
TABLE	MCtableItem	Purpose of the tabulated data	Table of text, numeric or datetime values.

4.14.3. Relationship Types between Content Items

The table below describes the Relationship Types between Source Content Items and the Target Content Items. The choice of which may be constrained by the IOD in which this Module is

contained. Merge DICOM Toolkit Definition column specifies the enumerated value used in the Toolkit to identify the Content Item Relationship.

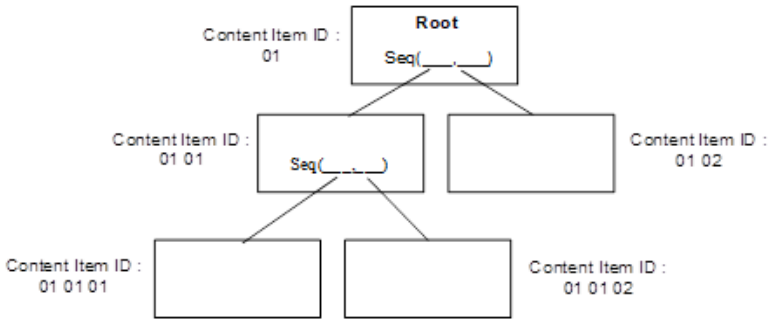
Table 4.11: SR Relationship Types

Relationship Type	Merge DICOM Toolkit Definition	Description
CONTAINS	SR_REL_CONTAINS	Source Item contains Target Content Item. For example: CONTAINER "History" {CONTAINS: TEXT: "mother had breast cancer"; CONTAINS IMAGE 36}
HAS OBS CONTEXT	SR_REL_HAS_OBS_CONTEXT	Has Observation Context. Target Content Items shall convey any specialization of Observation Context needed for unambiguous documentation of the Source Content Item. For example: CONTAINER: "Report" {HAS OBS CONTEXT: PNAME: "Recording Observer" = "Smith^John^^Dr^"}
HAS CONCEPT MOD	SR_REL_HAS_CONCEPT_MOD	Has Concept Modifier. Used to qualify or describe the Concept Name of the Source Content item, such as to create a post-coordinated description of a concept, or to further describe a concept. For example: CODE "Chest X-Ray" {HAS CONCEPT MOD: CODE "View = PA and Lateral"} For example: CODE "Breast" {HAS CONCEPT MOD: TEXT "French Translation" = "Sein"} For example: CODE "2VCXRPALAT" {HAS CONCEPT MOD: TEXT "Further Explanation" = "Chest X-Ray, Two Views, Posteroanterior and Lateral"}
HAS PROPERTIES	SR_REL_HAS_PROPERTIES	Description of properties of the Source Content Item. For example: CODE "Mass" {HAS PROPERTIES: CODE "anatomic location", HAS PROPERTIES: CODE "diameter", HAS PROPERTIES: CODE "margin", ...}.
HAS ACQ CONTEXT	SR_REL_HAS_ACQ_CONTEXT	Has Acquisition Context. The Target Content Item describes the conditions present during data acquisition of the Source Content Item. For example: IMAGE 36 {HAS ACQ CONTEXT: CODE "contrast agent", HAS ACQ CONTEXT: CODE "position of imaging subject", ...}.
INFERRED FROM	SR_REL_INFERRED_FROM	Source Content Item conveys a measurement or other inference made from the Target Content Items. Denotes the supporting evidence for a measurement or judgment. For example: CODE "Malignancy" {INFERRED FROM: CODE "Mass", INFERRED FROM: CODE "Lymphadenopathy",...}. For example: NUM: "BPD = 5mm" {INFERRED FROM: SCOORD}.

Relationship Type	Merge DICOM Toolkit Definition	Description
SELECTED FROM	SR_REL_SELECTED_FROM	Source Content Item conveys spatial or temporal coordinates selected from the Target Content Item(s). For example: SCOORD: "CLOSED 1,1 5,10" {SELECTED FROM: IMAGE 36}. For example: TCOORD: "SEGMENT 60-200mS" {SELECTED FROM: WAVEFORM}.

4.14.4. Content Item Identifier

Content Items are identified by their position in the Content Item tree. They have an implicit order as defined by the order of the Sequence Items. When a Content Item is the target of a by reference relationship, its position is specified as the Referenced Content Item Identifier in the source Content Item. The figure below illustrates an SR content tree and identifiers associated with each Content Item:



4.14.5. Observation Context

Observation Context describes who or what is performing the interpretation, whether the examination of evidence is direct or quoted, what procedure generated the evidence that is being interpreted, and who or what is the subject of the evidence that is being interpreted.

Initial Observation Context is defined outside the SR Document Content tree by other modules in the SR IOD (i.e., Patient Module, Specimen Identification, General Study, Patient Study, SR Document Series, Frame of Reference, Synchronization, General Equipment and SR Document General modules). Observation Context defined by attributes in these modules applies to all Content Items in the SR Document Content tree and need not be explicitly coded in the tree. The initial Observation Context from outside the tree can be explicitly replaced.

If a Content Item in the SR Document Content tree has Observation Context different from the context already encoded elsewhere in the IOD, the context information applying to that Content Item shall be encoded as child nodes of the Content Item in the tree using the HAS OBS CONTEXT relationship. That is, Observation Context is a property of its parent Content Item.

The context information specified in the Observation Context child nodes (i.e. target of the HAS OBS CONTEXT relationship) adds to the Observation Context of their parent node Content item, and shall apply to all the by-value descendant nodes of that parent node regardless of the relationship type between the parent and the descendant nodes. Observation Context is encoded in

the same manner as any other Content Item. Observation Context shall not be inherited across by-reference relationships.

Observation DateTime is not included as part of the HAS OBS CONTEXT relationship, and therefore is not inherited along with other Observation Context. The Observation DateTime Attribute is included in each Content Item which allows different observation dates and times to be attached to different Content Items.

The IOD may specify restrictions on Content Items and Relationship Types that also constrain the flexibility with which Observation Context may be described.

The IOD may specify Templates that offer or restrict patterns and content in Observation Context.

4.14.6. Structured Reporting Templates

Templates are patterns that specify the Concept Names, Requirements, Conditions, Value Types, Value Multiplicity, Value Set restrictions, Relationship Types and other attributes of Content Items for a particular application. SR Document templates are defined in the Part 16 of the DICOM Standard. Part 17 of the DICOM also has some explanatory information on encoding SR Documents. The Merge DICOM Toolkit SR Functions follow DICOM Templates structures and allow straightforward encoding based on template tables.

SR Templates are described using tables of the form shown in the table below.

Table 4.12: SR Template Definition

	NL	Rel with Parent	VT	Concept Name	VM	Req Type	Condition	Value Set Constraint
1								
2								
3								

a. Row Number

Each row of a Template Table is denoted by a row number. The first row is numbered 1 and subsequent rows are numbered in ascending order with increments of 1. This number denotes a row for convenient description as well as reference in conditions. The Row Number of a Content Item in a Template may or may not be the same as the ordinal position of the corresponding node in the encoded document. The Merge DICOM Toolkit does not use this number in any way.

b. Nesting Level (NL)

The nesting level of Content Items is denoted by ">" symbols, one per level of nesting below the initial Source Content Item (of the Template) in a manner similar to the depiction of nested Sequences of Items in Module Tables in Part 3 of the DICOM standard. When it is necessary to specify the Target Content Item(s) of a relationship, they are specified in the row(s) immediately following the corresponding Source Content Item. The Merge DICOM Toolkit provides functions to add nested (child) Content Items to the parent Content Item node.

c. Relationship with Source Content Item (Parent)

Relationship Type and Mode are specified for each row that specifies a target content item. The Relationship Types are enumerated in TABLE 4.11: SR RELATIONSHIP TYPES ON PAGE 83.

Relationship Type and Mode may also be specified when another Template is included, either "top-down" or "bottom-up" or both (i.e., in the "INCLUDE Template" row of the calling Template or in all rows of the included Template or in both places). There shall be no conflict between the Relationship Type and Mode of a row that includes another Template and the Relationship Type and Mode of the rows of the included Template.

When the relationship is defined in a form as R-RTYPE, it means that Relationship Mode is "By-reference" and Relationship Type is "RTYPE" (e.g., "R INFERRED FROM").

d. Value Type (VT)

The Value Type field specifies the SR Value Type of the Content Item or conveys the word "INCLUDE" to indicate that another Template is to be included (substituted for the row). The Merge DICOM Toolkit uses explicit function call for each Content Item Type as is described above.

e. Concept Name

Any constraints on Concept Name are specified in this field as defined or enumerated coded entries or as baseline or defined context groups. Alternatively, when the VT field is "INCLUDE", the Concept Name field specifies the template to be included.

Abbreviations used in templates

The following abbreviations are used in template definitions:

- **EV** Enumerated Value — values for are provided in the brackets.
- **DT** Defined Term — values are provided in the brackets.
- **BCID** Baseline Context Group ID — identifier that specifies the suggested Context Group. The suggested values can be found in DICOM Part 16 and identified by a Context ID provided in the brackets.
- **DCID** Defined Context Group ID — identifier that specifies the Context Group for a Coded Value that shall be used. The values can be found in DICOM Part 16 and identified by a Context ID provided in the brackets.
- **BTID** Baseline Template ID — identifier that specifies a template suggested to be used in the creation of a set of Content Items. The referenced template can be found in DICOM Part 16 and identified by a Template ID provided in the brackets.
- **DTID** Defined Template ID — identifier that specifies a template that shall be used in the creation of a set of Content Items. The referenced template can be found in DICOM Part 16 and identified by a Template ID provided in the brackets.

f. Value Multiplicity (VM)

The VM field indicates the number of times that either a Content Item of the specified pattern or an included Template may appear in this position. The table below specifies the values that are permitted in this field.

Table 4.13: Permitted Values for VM

Expression	Definition
i (where 'i' represents an integer)	Exactly i occurrences, where i >= 1. E.g. when i == 1 there shall be one occurrence of the Content Item in this position.
i—j	From i to j occurrences, where i and j are >= 1 and j > i.
1—n	One or more occurrences.

g. Requirement Type

The Requirement Type field specifies the requirements on the presence or absence of the Content Item or included Template. The following symbols are used:

- **M** - Mandatory. Shall be present.
- **MC** - Mandatory Conditional. Shall be present if the specified condition is satisfied.
- **U** - User Option. May or may not be present.
- **UC** - User Option Conditional. May not be present. May be present according to the specified condition.

h. Condition

The Condition field specifies any conditions upon which the presence or absence of the Content Item or its values depends. This field specifies any Concept Name(s) or Values upon which there are dependencies.

References may also be made to row numbers (e.g. to specify exclusive OR conditions that span multiple rows of a Template table).

The following abbreviations are used:

- **XOR** - Exclusive OR. One and only one row shall be selected from mutually exclusive options.

NOTE: For example, if one of rows 1, 2, 3 or 4 may be included, then for row 2, the abbreviation "XOR rows 1,3,4" is specified for the condition.

- **IF** - Shall be present if the condition is TRUE; may be present otherwise.
- **IFF** - If and only if . Shall be present if the condition is TRUE; shall not be present otherwise.
- **CV** - Code Value
- **CSD** - Coding Scheme Designator
- **CM** - Code Meaning
- **CSV** - Coding Scheme Version

i. Value Set Constraint

Value Set Constraints, if any, are specified in this field as defined or enumerated coded entries, or as baseline or defined context groups.

The Value Set Constraint column may specify a default value for the Content Item if the Content Item is not present, either as a fixed value, or by reference to another Content Item, or by reference to an Attribute from the dataset other than within the Content Sequence (0040,A730).

j. Inclusion of Templates

A Template may include another Template by specifying "INCLUDE" in the Value Type field and the identifier of the included Template in the Concept Name field. All of the rows of the specified Template are included in the invoking Template, effectively substituting the specified template for the row where the inclusion is invoked. Whether or not the inclusion is user optional, mandatory or conditional is specified in the Requirement and Condition fields. The number of times the included Template may be repeated is specified in the VM field.

We recommend that you implement templates as a subroutine or function call. In that case, the inclusion of the template will be implemented as a call to that template with passing parameters. Some of the templates defined in DICOM Part 16 already have predefined parameters and they are indicated by a name beginning with the character "$".

4.14.7. Memory Management

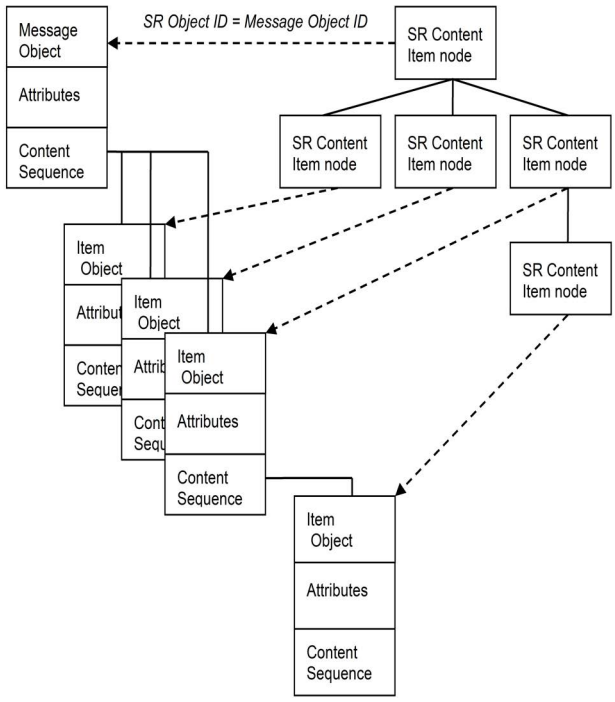
The Structured Reporting API is designed in such way that you only deal with types of objects:

- Message objects which are messages and message items.
- Structured Report objects which are the root SR Content Item and child Content Items.

You can convert back and forth between these objects and work with one object type at a time. However, it is imported to know how these objects are managed internally.

Structured Report Content Items are represented as a special SR objects in the memory. Each object is represented by the integer object ID that is also called a Node ID. The ID value for SR Objects is the same as the item ID or Message ID of the underlying message object. SR objects are always mapped to the message objects and use them as attribute storage. This allows you to use SR object ID in the basic toolkit functions like `MCattributeSet.setValue()`. Deletion of the SR

Object does not delete the underlying item object. The figure below shows the relationship between the SR tree and the message object:



When a message or item object is mapped with the SR object, it is marked with a special flag that prohibits some of the operations that can break the structured report hierarchy. For example, `MCattributeSet.duplicate()` will not work on the messages mapped with the SR object.

4.14.8. Overview of the Merge DICOM Toolkit SR Methods

The Merge DICOM Toolkit has several types of methods that can be used for reading/writing Structured Report IODs.

Low-level attribute access methods — These are the same methods that are used to work with attributes in message objects. Every SR Content Item ID is mapped to the Message Item ID and can be used to set or get additional attributes that are not covered by the High Level API.

Low-level navigation and conversion methods — These methods provide mapping and conversion between message objects and SR objects (Content Items). The following methods are included:

- `MCstructuredReport()` — initialize a new SR with SR root object and maps it to the existing message object.
- `MCstructuredReport.read(MCattributeSet)` — transform Structured Report attribute set into tree data structure
- `MCstructuredReport.write(MCattributeSet)` — structured Report document from a tree data structure to DICOM attribute set.
- `MCstructuredReport.createItem()` — creates an SR Content Item and maps it to the existing message item.

- `MCstructuredReport.findItem(MCitemIdentifier)` — retrieve the child Content Item specified given `MCitemIdentifier` item Id.

High-Level methods for encoding SR - These methods can be used to build an entire SR document tree with minimum coding. The following methods are included:

- `MCcontentItem.addChild(MCcontentItem, relation)` — add a new Content Item child, for example of type "TEXT" and 'relation' specifies an `MCrelationType` relationship between the enclosing source Content Item and the target Content Item.
- `MCcontentItem.removeChild(MCcontentItem)` — remove Content Item child.
- `MCcontentItem.addReference(reference)` — create a `MCitemRelation` reference to another Content Item node "by reference".
- `MCcontentItem.removeReference(MCcontentItem)` — remove a reference to another to another Content Item node.

4.14.9. Encoding SR Documents

The creation of the SR document involves the following steps:

1. Create a new SR object as a root node.
2. Add Content Items (nodes) to the tree based on the template definitions.
3. Convert SR object to a message object.
4. Add Patient/Study/Series and other attributes required by the IOD definition,
5. Save the result message object to a file.

To create a new SR, you need to know the IOD type you are creating and the templates that will be used to generate the SR Document Content.

a. Key Object Selection Example

The Key Object Selection document is constrained by a single template. The following template is taken from DICOM Part 16.

TID 2010
KEY OBJECT SELECTION
Type: Non-Extensible

	NL	Rel with Parent	VT	Concept Name	VM	Req Type	Condition	Value Set Constraint
1	>		CONTAINER	DCID(7010) Key Object Selection Document Titles	1	M		Root node
2	>	HAS CONCEPT MOD	CODE	EV (113011, DCM,"Document Title Modifier")	1-n	U		

	NL	Rel with Parent	VT	Concept Name	VM	Req Type	Condition	Value Set Constraint
3	>	HAS CONCEPT MOD	CODE	EV (113011, DCM,"Document Title Modifier")	1	UC	IF Row 1 Concept Name = (113001, DCM, "Rejected for Quality Reasons") or (113010, DCM," Quality Issue")	DCID (7011)
4	>	HAS CONCEPT MOD	CODE	EV (113011, DCM,"Document Title Modifier")	1	MC	IF Row 1 Concept Name = (113013, DCM, "Best In Set")	DCID (7012)
5	>	HAS CONCEPT MOD	INCLUDE	DTID(1204) Language of Content Item and Descendants	1	U		
6	>	HAS OBS CONTEXT	INCLUDE	DTID(1002) Observer Context	1-n	U		
7	>	CONTAINS	TEXT	EV(113012, DCM,"Key Object Description")	1	U		
8	>	CONTAINS	IMAGE	Purpose of Reference shall not be present	1-n	MC	At least one of Rows 8, 9 and 10 shall be present	
9	>	CONTAINS	WAVEFORM	Purpose of Reference shall not be present	1-n	MC	At least one of Rows 8, 9 and 10 shall be present	
10	>	CONTAINS	COMPOSITE	Purpose of Reference shall not be present	1-n	MC	At least one of Rows 8, 9 and 10 shall be present	

The code below generates a valid DICOM KO object and illustrates how the template is encoded using the Merge DICOM Toolkit functions.

```
#

# Create OB-GYN ULTRASOUND REPORT.

# Note: This SR does not follow the DICOM template, but only some
portion of it.

#

sr = MCstructuredReport('5000', MCcontainerItem.SEPARATE,
MCcodedEntry('125000', 'DCM', 'OB-GYN Ultrasound Proc)

root = sr.getRoot()


#

# IMAGE LIBRARY
```

```
#
citem = MCcontainerItem(None, MCcontainerItem.SEPARATE,
MCcodedEntry('111028', 'DCM', 'Image Library'))
root.addChild(citem, MCrelationType.CONTAINS)

item = MCcompositeItem(MCsopInstanceReference('1.2.3.4.2',
'1.2.3.4.5.2'))
citem.addChild(item, MCrelationType.CONTAINS)

item = MCimageItem(CsopInstanceReference('1.2.3.4.1', '1.2.3.4.5.1'))
citem.addChild(item, MCrelationType.CONTAINS)
item.setFrames([1, 2, 3, 4, 5])

item = MCwaveformItem( MCsopInstanceReference('1.2.3.4.3',
'1.2.3.4.5.3'))
citem.addChild(item, MCrelationType.CONTAINS)
item.setChannels([(1,2), (3,4), (5,6)])

#
# PATIENT CHARACTERISTICS
#
citem = MCcontainerItem(None, MCcontainerItem.SEPARATE,
MCcodedEntry('121118', 'DCM', 'Patient Characteristics'))
root.addChild(citem, MCrelationType.CONTAINS)

item = MCtextItem('Patient looks ok', MCcodedEntry('121106', 'DCM',
'Comment'))
item.setDateTime(MCdateTime(MCdate(2000, 10, 10), MCtime(20,30,40)))
citem.addChild(item, MCrelationType.CONTAINS)

item = MCnumItem('1.85', MCcodedEntry('m', 'LN', 'meters'),
MCcodedEntry('8302-2', 'LN', 'Patient Height'))
citem.addChild(item, MCrelationType.CONTAINS)
```

4.14.10.Reading SR Documents

Reading SR Documents is done in a similar way as encoding, but in reverse sequence:

- Read a File or receive a message object.
- Read root level attributes.

- Convert message object into SR object.
- Traverse SR content tree and extract Content Node attributes,

The following code demonstrates a reading sequence for the Key Object Document generated above.

```
file = MCfile()
file.readP10File(fname)

sr = MCstructuredReport()
sr.read(file)

root = sr.getRoot()

dst = MCfile()
sr.write(dst)

dst.writeP10File(fname)
dst.dispose()
```

4.15. Converting Attribute Set to/from DICOM JSON Model String

An attribute set can be converted to a DICOM JSON Model string by using the **writeToJSON** method of the MCattributeSet class. The writeToJSON method creates a DICOM JSON Model (PS3.18) string describing the contents of the attribute set. The JSON output is written to the stream identified by the stream object provided.

The attribute values can be read from a DICOM JSON Model string into an MCattributeSet object using the **readFromJSON** method of MCattributeSet.

4.16. Converting Attribute Set to/from Native DICOM Model XML String

An attribute set can be converted to a Native DICOM Model XML string by using the **writeToXMLNative** method of the MCattributeSet. The writeToXMLNative method creates a Native DICOM Model (PS3.18) XML string describing the contents of the attribute set. The XML output is written to the stream identified by the stream object provided.

The attribute values can be read from a Native DICOM Model XML string into an MCattributeSet object using the **readFromXMLNative** method of MCattributeSet.

Chapter 5. Deploying Applications

There are several issues to consider when deploying a Merge DICOM Toolkit based application. These include deciding which Merge DICOM Toolkit files are needed for your application, how to set important configuration options to reduce problems in the field, and how to deal with potential UN VR problems. The following sections describe these issues in further detail.

5.1. Merge DICOM Toolkit Required Files

There are a limited number of files required by Merge DICOM Toolkit applications. These files are described in the table below. Note that the use of some of these files can be avoided by using the `genconf` and `gendict` utilities. Each of these utilities generates a source file from the configuration files that can then be compiled and linked into your application.

Table 5.1: Files needed when deploying an application

File	Description and Use
merge.ini	Merge DICOM Toolkit initialization file. This file contains logging information and path names for the other configuration files. Use of this file can be avoided by using the `genconf` utility to link the file into the Merge DICOM Toolkit application.
mergecom.pro	Merge DICOM Toolkit system profile. This file contains general run-time configuration options. Use of this file can be avoided by using the `genconf` utility to link it into the Merge DICOM Toolkit application.
mergecom.app	Merge DICOM Toolkit application profile. This file contains configuration information about the services supported by the Merge DICOM Toolkit application and information about remote DICOM applications. Use of this file can be avoided by using the `genconf` utility to link it into the Merge DICOM Toolkit application.
mergecom.srv	Merge DICOM Toolkit services file. This file contains information about the services supported by Merge DICOM Toolkit. Use of this file can be avoided by using the `genconf` utility to link it into the Merge DICOM Toolkit application.
mrgcom3.msg	Merge DICOM Toolkit message information file. This file contains validation information for DICOM messages.
mrgcom3.dct	Merge DICOM Toolkit data dictionary file. This file contains information about all of the DICOM attributes. Use of this file can be avoided by using the `gendict` utility to link it into the Merge DICOM Toolkit application.

5.2. Configuration Options

The majority of Merge DICOM Toolkit's configuration options can be used to solve interoperability problems in the field. There are some options, however, that can be set before deploying a Merge

DICOM Toolkit application to help reduce potential problems. These options are listed in the table below with descriptions of how they can be set.

Table 5.2: Configuration options to consider when deploying an application

Configuration Option	Description
ACCEPT_ANY_APPLICATION_TITLE	When set to NO, Merge DICOM Toolkit requires that the Application Entity title sent in an association request match one of the registered application titles for the SCP. When there is no match, the association will be automatically rejected. Setting this option to YES will eliminate some association negotiation problems in the field for SCP applications.
ACCEPT_ANY_HOSTNAME	When set to NO, Merge DICOM Toolkit will attempt to resolve the IP address of the SCU application into a hostname. If this resolution cannot be done, the association will automatically be rejected. Setting this option to YES will reduce configuration problems in the field for SCP applications.
EXPORT_UN_VR_TO_MEDIA	Setting this option to NO will cause UN VR attributes to not be exported when writing DICOM Part 10 format files. See the following sections for a further discussion of UN VR.
EXPORT_UN_VR_TO_NETWORK	Setting this option to NO will cause UN VR attributes to not be exported over the network. See the following sections for a further discussion of UN VR.
IMPLEMENTATION_CLASS_UID	The Implementation Class UID is used to uniquely identify a specific class of implementation. PS3.7 of DICOM states: "(The Implementation Class UID) is intended to provide respective (each network node knows the other's implementation identity) and non-ambiguous identification in the event of communication problems encountered between two nodes." PS3.7 of DICOM further defines how this UID should be defined: "different equipment of the same type or product line (but having different serial numbers) shall use the same Implementation Class UID if they share the same implementation environment (that is, software)."
IMPLEMENTATION_VERSION	The Implementation Version is intended to distinguish between software versions of an implementation. It should be set to the version of the Merge DICOM Toolkit application.

5.3. UN VR

DICOM Supplement 14, Unknown Value Representation, became a part of the DICOM standard on June 3, 1997. This supplement adds a new value representation, UN, to the DICOM standard. It was developed to fix two related holes in the DICOM standard:

When standard or private attributes were received in an implicit value representation (VR) transfer syntax, and the user does not have a knowledge of the VR of the attributes, there is no way to represent the VR for these attributes in an explicit VR transfer syntax.

Every time a new VR is added to the standard, there is no way to determine if the length field in explicit value representation transfer syntaxes should be encoded as 2 bytes or 4 bytes, so a general parser could not be properly written to handle future VRs.

The need for this supplement is mainly for use in "archive" systems. An "archive" will typically want to preserve the private attributes contained within a message for later use. There also may be a need to add support for new image objects with new VRs to an "archive" system without having to change the software.

Unfortunately, the method that Supplement 14 specifies for encoding UN value representation attributes is typically not compatible with older DICOM implementations. Versions previous to 2.2.2 of the Merge DICOM Toolkit do not parse these attributes properly. The `MCassociation.read()` method will fail and the association will be aborted if a UN VR attribute is received. This has obviously caused a variety of interoperability problems in the field.

The typical DICOM scenario where UN VR can cause a DICOM communication failure is the following: a modality exports a series of images to a PACS or "archive" system via the DICOM storage service class. The images were encoded in the implicit VR little endian transfer syntax and contain multiple private attributes. Later, a DICOM workstation decides to retrieve the images from the "archive" or PACS system. The workstation does not yet support UN VR, however, the PACS or "archive" system does. The workstation uses the DICOM query/retrieve service class to retrieve the series of images. When the images are exported to the workstation with an explicit VR transfer syntax, the workstation fails to parse the first image received when it encounters the first UN VR attribute, and the association is automatically aborted by the workstation.

We have added several methods to solve this interoperability problem through the Merge DICOM Toolkit's configuration files. For SCU systems that are exporting UN VR tags to systems that cannot handle them, the following can be done:

Configure the SCU to only use the Implicit VR Little Endian transfer syntax when exporting objects. This can be done through the use of transfer syntax lists within the `mergecom.app` file or through commenting out the UID definitions for the other transfer syntaxes within the mergecom.pro file.

Set the `UNKNOWN_VR_CODE` configuration option in the mergecom.pro file to 'OB'. This forces unknown VR attributes to be encoded as OB instead of as UN. All implementations can handle OB encoding. There are several drawbacks to this option. If the attributes are encoded as OB, it is harder for these attributes to be converted back to their normal VR. Secondly, this option changes all instances of the UN VR into OB. Systems that can handle the UN VR will now also receive these attributes as OB.

Set the `EXPORT_UN_VR_TO_NETWORK` configuration option to 'No'. This will cause the Merge DICOM Toolkit to not export attributes encoded as UN VR to the network. This option was added to release 2.3.0 of the Merge DICOM Toolkit.

For SCP systems receiving UN VR tags when they cannot handle them, the following can be done:

Configure the SCP to only negotiate the Implicit VR Little Endian transfer syntax when receiving objects.

With the help of these options, most UN VR problems in the field can be fixed simply by changing configuration values with the Merge DICOM Toolkit.

Appendix A. Frequently Asked Questions

This appendix lists some frequently asked questions by Merge DICOM Toolkit users.

1. *I recently received a new version of Merge DICOM Toolkit and wonder what is required for me to upgrade to the new version?*

   There are several areas where changes typically occur between releases of the Merge DICOM Toolkit. The following are specific areas to look at when upgrading to a new version:

   - **Upgrading the library** — The Merge DICOM Toolkit library itself must be updated by replacing the library.
   - **Upgrading the data dictionary files** — The mergecom.srv, mrgcom3.msg, and mrgcom3.dct files must all be updated when upgrading the Merge DICOM Toolkit data dictionary. Upgrading some, but not all of these files can cause subsequent problems.
   - **Upgrading the configuration files** — Upgrading the merge.ini, mergecom.pro, and mergecom.app configuration files is optional. Although new configuration options are often added to these files, Merge DICOM Toolkit will assume default values for these options if they are not included in a configuration file. These files do not have to be updated when moving to a new release. Note however that the descriptions of configuration options are often updated and it is useful to have the latest versions of these files.

2. *I am running the toolkit's sample applications for the first time. I have set the MERGE_INI environment variable to point to the merge.ini file. However, the* `MC.initialize()` *is still throwing an exception with* ***`MC_CONFIG_INFO_ERROR`*** *id. What is the cause of this problem?*

   This is usually only a problem under Windows. The merge.ini file contains several entries that point to the locations of the other toolkit configuration files. These entries contain relative pathnames for the other files. If the sample applications are not executed from the directory where the configuration files are located, the toolkit will be unable to find the files and produce this error. Changing these paths to absolute paths will fix the problem.

3. *It is inconvenient to set absolute paths for the various configuration options in the merge.ini and mergecom.pro files that need them. Is there a way to make these pathnames be configurable at run-time?*

   Merge DICOM Toolkit allows the placement of environment variables in these pathnames. This allows setting of a root directory for these pathnames. The following is an example of how this functionality is used in our configuration files:

   - MERGECOM_PRO = $(MERGE_ROOT)\mc3apps\mergecom.pro
   - In this example, MERGE_ROOT would be an environment variable set in a similar fashion as the MERGE_INI environment variable.
   - A special macro "MC3INIDIR" is used to represent the directory where "merge.ini" is. It is used like the environment variable with the difference that it is automatically resolved and does not need to be set.
   - If MERGECOM_3_PROFILE, MERGECOM_3_SERVICES or MERGECOM_3_APPLICATIONS contain relative paths with a prefix "$(MC3INIDIR)" or "%MC3INIDIR%", the toolkit considers the path relative to the location of the "merge.ini" file.

   For example:

   ```
   MERGECOM_3_PROFILE = $(MC3INIDIR)../config/mergecom.pro
   ```

The path of the profile file is "../config/mergecom.pro" relative to the location of the "merge.ini" file.

4. *I am testing the sample applications for the first time and cannot get the client (SCU) application to connect to the server (SCP) for any of the sample applications. The* `MCassociation.requestAssociation()` *method is throwing an exception. It appears as though the connection is opening, but it is quickly dropped. Why is this happening?*

   As a security measure, the SCPs association process call attempts to determine the hostname of SCUs connecting to it. If it cannot determine the remote hostname, it will drop the connection. The SCPs association process uses the local system's hosts file or its configured domain name server to translate the SCU's IP address into its hostname. By configuring the SCU's hostname in your local hosts file, this problem will be eliminated. Also, the **ACCEPT_ANY_HOSTNAME** configuration value in the mergecom.pro file disables this checking.

5. *What can be done to reduce the memory requirements of the Merge DICOM Toolkit?*

   There are several configuration values that reduce Merge DICOM Toolkit's memory requirements. The following describes each of these options:

   - **FORCE_OPEN_EMPTY_ITEM** — It is especially useful for reducing the amount of memory used when working with large DICOMDIRs.
   - **LARGE_DATA_STORE** and **LARGE_DATA_SIZE** — These options control the ability of Merge DICOM Toolkit to store pixel data in temporary files instead of RAM. This functionality is enabled by setting `LARGE_DATA_STORE to FILE`, and adjusting `LARGE_DATA_SIZE` to the size of data element that you want spooled to temporary file. Note that this will decrease performance.
   - **DICOMDIR_STREAM_STORAGE** — This option can be used when reading DICOMDIR files to reduce the amount of memory required to store directory records within the DICOMDIR.

6. *What can be done to increase the performance of the Merge DICOM Toolkit?*

   There are several Merge DICOM Toolkit configuration values that impact performance in different ways. The following is a summary of these options:

   - **ELIMINATE_ITEM_REFERENCES** — This option will disable functionality within the toolkit that causes the toolkit to search all currently open message objects for references to an item that is being freed by one of these calls. This call is especially useful when your application uses very large DICOMDIR files.
   - **PDU_MAXIMUM_LENGTH** — This option sets the maximum sized PDU that the toolkit will receive. If during association negotiation the maximum sized PDU of the system negotiating with the toolkit application is larger than this value, the PDU size will be limited to this value.

     Setting this option so that a PDU fits within an even multiple of the default TCP/IP Maximum Segment Size (MSS) of 1460 bytes will increase performance. Note that 6 bytes for the PDU header must be added to the configured maximum PDU size when calculating a multiple of the MSS. Having the PDU Maximum length an even multiple of the MSS ensures that there are limited delays within TCP/IP stack when transferring. With the exception of the final TCP/IP packet for a message, all packets transferred should exactly fit within a TCP/IP packet.
   - **WORK_BUFFER_SIZE** — This option specifies how the toolkit buffers data before storing it or passing it to a user callback function. Setting higher values for this option will increase performance.
   - **TCPIP_RECEIVE_BUFFER_SIZE** — This option sets the TCP/IP receive buffer size. Higher values for this buffer generally will increase the network performance of the toolkit for server (SCP) applications. This value should also be slightly larger than the

PDU_MAXIMUM_LENGTH to increase performance. Setting this value to an even multiple of the MSS (1460 bytes) will help increase performance on most platforms.

- **TCPIP_SEND_BUFFER_SIZE** — This option sets the TCP/IP send buffer size. Higher values for this buffer generally will increase the network performance of the toolkit for client (SCU) applications. This value should also be slightly larger than the **PDU_MAXIMUM_LENGTH** to increase performance. Setting this value to an even multiple of the MSS (1460 bytes) will help increase performance on most platforms.
- **EXPORT_UNDEFINED_LENGTH_SQ** — This option determines how Merge DICOM Toolkit encodes sequences within all non-DICOMDIR messages and files. When set to Yes, the sequences are encoded as undefined length. This eliminates the need for Merge DICOM Toolkit to determine the length of sequences and increases performance.
- **EXPORT_GROUP_LENGTHS_TO_NETWORK** — This option determines if Merge DICOM Toolkit encodes group length attributes when writing to the network (if they are included in the message being sent). Setting this option to No increases Merge DICOM Toolkit network performance. This eliminates the need for Merge DICOM Toolkit to determine the length of groups when streaming to the network.
- **EXPORT_GROUP_LENGTHS_TO_MEDIA** — This option determines if Merge DICOM Toolkit encodes group length attributes when writing to files. Setting this option to No increases Merge DICOM Toolkit performance. This eliminates the need for Merge DICOM Toolkit to determine the length of groups when writing to media.
- **EXPORT_UNDEFINED_LENGTH_SQ_IN_DICOMDIR** — This option determines how Merge DICOM Toolkit exports sequence attributes in DICOMDIRs. When set to Yes, the sequences in DICOMDIRs are encoded as undefined length. This greatly improves performance when writing DICOMDIRs because Merge DICOM Toolkit no longer needs to calculate the length of sequence attributes in DICOMDIRs.

7. *Which of the options listed above have the greatest impact on network performance?*

   - The **PDU_MAXIMUM_LENGTH**, **TCPIP_RECEIVE_BUFFER_SIZE** and **TCPIP_SEND_BUFFER_SIZE** configuration options have the greatest impact on network performance. Setting these to higher values directly increases the network performance of Merge DICOM Toolkit.
   - **EXPORT_UNDEFINED_LENGTH_SQ** can have a large impact if many sequence attributes are included in the message being transferred.

8. *I am sending 8-bit images with Merge DICOM Toolkit, however, after sending the data to another system, the pixel data is byte swapped incorrectly. What is causing this problem?*

   The Merge DICOM Toolkit User's Manual contains the section "8-bit Pixel Data" which describes this problem. This is typically only a problem on Big Endian machines. To summarize the problem, we expect 8-bit data to be byte swapped on big endian machines. We do not look at the "bits allocated" and "bits stored" tags to determine that the pixel data itself is 8-bit data, we always treat pixel data (`7fe0,0010`) as OW.

9. *I recently upgraded to a new release of the Merge DICOM Toolkit. Since this upgrade, I have been having problems with `MCattributeSet.setValue()` method throws an exception with `MC_INVALID_TAG` id. This code worked before the upgrade. What is causing these problems?*

   The Merge DICOM Toolkit data dictionary changes from release to release. In some cases, the identification number for a particular message type changes. When upgrading, if you do not change all of the data dictionary files, this error will occur. The following files should be upgraded with each release:

   - mergecom.srv

- mrgcom3.msg
- mrgcom3.dct

10. *What are the differences between the `MC_NULL_VALUE`, `MC_EMPTY_VALUE` and `MC_INVALID_TAG` ?*
    - The `MC_NULL_VALUE` return value is used to identify when an attribute within a DICOM message has zero length. DICOM allows attributes that have a Value Type of 2 to be set to zero length when their value is unknown.
    - The `MC_EMPTY_VALUE` and `MC_INVALID_TAG` return values both mean that a message does not contain a value for the specified attribute. The use of these return values depends on how the message, file, or item containing the attribute was created.
    - Merge DICOM Toolkit loads a list of all of the valid attributes for the new created object. For these types of attributes, the `MCattributeSet.getValue()` functions will throw an exception with `MC_EMPTY_VALUE` id when an attribute defined for the object does not have a value. They will throw an exception `MC_INVALID_TAG` id for attributes that are not defined for the object.
11. *I am trying to assign the value to a DICOM attribute within a message, but Merge DICOM Toolkit will not allow me to do this. When I call the `MCattributeSet.setValue()`, they are throwing an exception with `MC_INVALID_TAG` id. How can I add this attribute?*
    - This problem occurs when Toolkit creates a message or file of a particular type. These functions restrict the attributes that can be added to a message. Only those attributes that have been defined for the message type (and can be found in our `message.txt` file) can be assigned to the message or file.
    - When adding an attribute that has not been defined for a message, `MCattributeSet.addValue()` can be called to add the tag to the definition of the message. Subsequent calls to the `MCattributeSet.setValue()` functions will then allow the user to assign the attribute.

Appendix B. Unique Identifiers (UIDs)

UIDs provide the capability to identify many different types of items. The purpose of UIDs is to guarantee the uniqueness of these types of items. DICOM uses UIDs to uniquely identify items such as SOP classes, image instances and network negotiation parameters. Part 5, Section 9 along with Annexes B and C of the DICOM Standard discusses how UIDs are composed, encoded and registered.

B.1. Summary of UID Composition

A UID is composed of a number of numeric values as defined by ISO 8824. The following is a typical example of a UID:

```
1.2.840.10008.2.45.1.12345
```

A UID is composed of two parts: a `<root>` and a `<suffix>` and has the following form:

```
UID = <root>.<suffix>
```

where `<root>` is assigned by a registration authority (e.g., ANSI) with the distinguishing component being the organization ID. The `<root>` portion of the UID uniquely identifies an organization while the `<suffix>` portion is used to uniquely identify a specific object within the scope of the organization. While the `<root>` component of the UID stays constant, the `<suffix>` portion will change in a manner that will provide uniqueness for objects that need UIDs. Note: this implies that the organization is responsible for maintaining the uniqueness of the `<suffix>`.

For example, using the UID above, `<root> = 1.2.840.10008` and `<suffix> = 2.45.1.12345`. Where the organization ID portion of the `<root>` (10008) distinguishes organizations from each other.

NOTE: The above example is typical for UIDs obtained by ANSI during the time when the DICOM standard was first released. The organization ID of 10008 has actually been assigned to NEMA and is used as part of the `<root>` for DICOM standard UIDs such as SOP Classes, Transfer Syntaxes, etc. For example, vendors creating images need to obtain their own organization ID and cannot use 10008.

For future UIDs, ISO has developed a joint relationship with CCITT and has changed the `<root>` structure. Therefore, new UIDs from ANSI will no longer be of the form 1.2.840.xxxxx. but are currently assigned using the form, `<root> = 2.16.840.1.10008`, where, of course, 10008 is the organization ID.

B.2. Sample UID Format

There are many methods that can be used to ensure the uniqueness of a UID. The following is one example encoding of a UID to ensure uniqueness:

```
<root>.<serial>.<process id>.<timestamp>.<count>
```

In this example, `<root>` is the assigned root UID for an organization. The `<serial>` component would be a unique serial number assigned to the product within the organization. It may also be an encoding of the MAC address assigned to an Ethernet card in the system. This field along with the root gives a base for the UIDs that is unique for a specific device. The remaining components ensure that the UID is unique within that device.

The <process id> field would be a process ID or thread ID for the process generating the UID. The <timestamp> field would be a timestamp generated when the process or thread is created. Finally, the <count> field would be a unique counter that is incremented for each UID created.

Some vendors also include fields within the UID to identify the type of UID. For example, the first component after the root within the UID may be a ".1" for Study Instance UIDs, a ".2" for Series Instance UIDs, and".3" for SOP Instance UIDs. Occasionally a product identifier will also be included within a UID. This may be a unique number assigned to a product within an organization. Finally, a number may also be added to signify the software revision number for a product that is generating the UID.

B.3. Obtaining a UID

The <root> portion of the UID should be registered by an organization that guarantees global uniqueness. The American National Standards Institute (ANSI) is the registration authority for the United States. Other national registration authorities exist for nations throughout the world such as IBN in Belgium, AFNOR in France, BSI in Great Britain, DIN in Germany, and COSIRA in Canada.

B.3.1. Obtaining a UID from ANSI

ANSI is the registration authority for the US for organization names (i.e. <root>) under the global registration process established by the International Standards Organization (ISO) and the International Telegraph and Telephone Consultative Committee (CCITT). ANSI's registration service conforms with CCITT X.660 and ISO/IEC 9834-1. The ANSI organization name registration service assigns one name component to the hierarchy defined by CCITT and ISO/IEC.

An organization seeking registration may do so by submitting a Request for Registration application form along with a fee (as of August 1996 the fee is $1,000) to the Registration Coordinator. The Request for Registration application form can be obtained from ANSI by use of the following information:

American National Standards Institute

11 West 42nd Street

New York, New York 10036

TEL: 212.642.4900 FAX: 212.398.0023

Appendix C. Writing a DICOM Conformance Statement

Detailed below is a guideline for writing a DICOM conformance statement for your application. Since the Toolkit is not an application, this section only gives an outline of the DICOM services it supports. Responsibility for full DICOM conformance to particular SOP classes rests with the application developer, since many of the requirements for such conformance lie outside the realm of the Toolkit. For example, the high level behavior of Query/Retrieve service class SCUs and SCPs as defined in Part 4 of the DICOM standard, is implemented by the application developer in conjunction with the toolkit functionality.

C.1. Conformance Statement Sections

C.1.1. Implementation Model

The Implementation model consists of three sections:

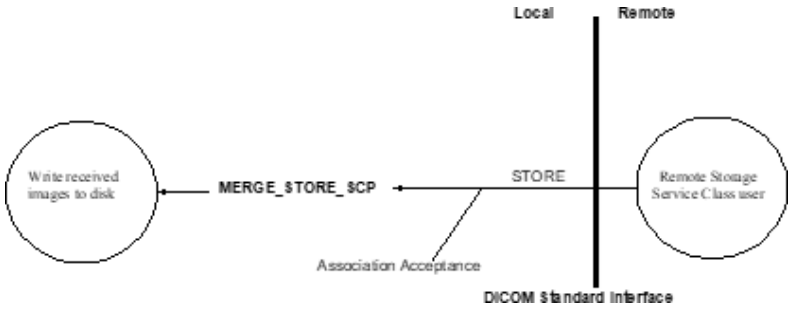
- the Application Data Flow Diagram which specifies the relationship between the Application Entities and the “external world” or Real-World activities.
- a functional description of each Application Entity.
- the sequencing constraints among them.

C.1.2. Application Data Flow

As part of the Implementation model, an Application Data Flow Diagram is included. This diagram represents all of the Application Entities present in an implementation, and graphically depicts the relationship of the AEs' use of DICOM to Real-World Activities as well as any applicable user interaction.

The Merge DICOM Toolkit provides the core functionality required to facilitate data flow between SCUs and SCPs.

Application conformance statements include a data flow diagram. An example is shown below for a simple Storage Service Class SCP.



a. Functional Definition of Application Entities (AE)

This section contains a functional definition for each individual, local Application Entity. It describes in general terms, the functions that are performed by the AE, and the DICOM services used to accomplish these functions. In this sense, "DICOM services" refers not only to DICOM Service Classes, but also to lower level DICOM services, such as Association Services.

Application conformance statements are described in this section with a general specification of functions to be performed by SCU or SCP.

C.1.3. Sequencing of Real World Activities

If applicable, this section will contain a description of sequencing as well as potential constraints on real-world activities. These include any applicable user interaction as performed by all the AEs. A UML sequence diagram that depicts the real-world activities as vertical bars, and shows events exchanged between them as arrows, is strongly recommended.

Application conformance statements are included in this section along with any associated sequence of real-world activities. For example, a Storage Service Class SCP might perform the following real- world activities: store an image, modify it in some defined manner, act as a Storage Service Class SCU and forward the modified image somewhere.

C.1.4. AE Specifications

The next section in the DICOM Conformance Statement is a set of Application Entity specifications. There is one specification for the AE. Each individual AE specification has a subsection. There are as many of these subsections as there are different AEs in the implementation. That is, if there are two distinct AEs, then there are two subsections. The Merge DICOM Toolkit uses the mergecom.app configuration file to read configuration parameters for each AE. The following subsections are filled in for each AE:

Application Entity

- SOP Classes
- Association Policies
    - General
    - Number of Associations
    - Asynchronous Nature
    - Implementation Identifying Information
- Association Initiation Policy
    - Activity
        - Description and Sequencing of Activities
        - Proposed Presentation Contexts
        - SOP Specific Conformance for SOP Class(es)
- Association Acceptance Policy
    - Activity
        - Description and sequencing of Activities
        - Accepted Presentation Contexts
        - SOP Specific Conformance for SOP Class(es)

C.1.5. SOP Classes

Application conformance statements specify the DICOM SOPs which are supported by each Application Entity. For SCP Entities, the initiation of associations. See D.3. SYSTEM PROFILE ON PAGE 125 and the `MCapplication.startListening()` API in the Python source code. For SCU Entities, the list of supported SOP classes will correspond to the services specified in "mergecom.app" for any SCPs to which the SCU wishes to connect.

C.1.6. Number of Associations

The Merge DICOM Toolkit does not impose any limit on the number of simultaneous associations that can be requested or accepted. The only limitation on the number of simultaneous associations is imposed by the operating system and available resources. However, if your application enforces this limit, it is defined here.

The MAX_PENDING_CONNECTIONS setting in the "mergecom.pro" file refers to the maximum number of outstanding connection requests per listener socket. It does not limit the maximum number of simultaneous associations.

C.1.7. Asynchronous Nature

Merge DICOM Toolkit does not currently support multiple outstanding transactions over a single association.

C.1.8. Implementation Identifying Information

Application conformance statements specify the Implementation Class Unique Identifier (UID) for the application, as well as the Implementation version name. These identifiers are taken from the mergecom.pro configuration file under the following keys:

```
IMPLEMENTATION_CLASS_UID

IMPLEMENTATION_VERSION
```

This UID must follow the syntax rules specified in Part 5 of the DICOM standard.

a. Proposed or Accepted Presentation Contexts

Application conformance statements specify all presentation contexts that are used for association negotiation. A presentation context consists of:

- an Abstract Syntax which is a DICOM service class name and unique identifier(UID);
- a transfer syntax name and UID. A transfer syntax represents a set of data encoding rules that are specified in the "mergecom.pro" file. See D.3. SYSTEM PROFILE ON PAGE 125.
- the role that the application will perform within the service class. The roles associated with a particular service class are discussed in Part 4 of the DICOM standard.
- any extended negotiation information used when creating associations. See the `MCassociation.getInfo()` method.
- any rules that govern the acceptance of presentation contexts for the AE. This includes rules for which combinations of Abstract/Transfer Syntaxes are acceptable, and rules for prioritization of presentation contexts. Rules that govern selection of transfer syntax within a presentation

context are stated here. See D.2. APPLICATION PROFILE ON PAGE 112. Also, see the `MCassociation.getInfo()` API to learn about the presentation contexts that are queryable by an application program.

Refer to the table below for an example.

Table C.1: Example Presentation Context

Presentation Context Table					
Abstract Syntax		**Transfer Syntax**		**Role**	**Extended**
Name	**UID**	**Name List**	**UID List**		**Negotiation**
Computed Radiography Image Storage	1.2.840.10008.5.1.4.1.1.1	DICOM Implicit VR Little Endian	1.2.840.10008.1.2	SCP	None
		DICOM Explicit VR Little Endian	1.2.840.10008.1.2.1		
		DICOM Explicit VR Big Endian	1.2.840.10008.1.2.2		

Merge DICOM Toolkit uses mergecom.app configuration settings to specify presentation contexts shown above.

C.1.9. SOP Specific Conformance

This section includes the SOP specific behavior, i.e., error codes, error and exception handling and time-outs, etc. The information is described in the SOP specific Conformance Statement section of PS 3.4 (or relevant private SOP definition).

C.1.10. Transfer Syntax Selection Policies

Merge DICOM Toolkit uses the following policy when selecting a transfer syntax:

- An SCU offers any transfer syntaxes which are defined in its mergecom.pro file.
- The SCP prefers it's native byte ordering, and will prefer explicit over implicit VR.

C.2. Network Interfaces

C.2.1. Physical Network Interface

Merge DICOM Toolkit runs over the TCP/IP protocol stack on any physical interconnection media supporting the TCP/IP stack.

C.2.2. IPv4 and IPv6 Support

Merge DICOM Toolkit supports both IPv4 and IPV6 protocols and is configurable in the system profile.

C.2.3. Configuration

Refer to APPENDIX D. CONFIGURATION PARAMETERS ON PAGE 109 for complete configuration information.

Applications reference four (4) configuration files. The first, merge.ini, is found through the MERGE_INI environment variable. They are as follows:

- merge.ini - Specifies the names of the other three (3) configuration files and also contains message logging parameters.
- mergecom.pro - Specifies run-time parameters for the application.
- mergecom.app - Defines service lists and applications on other network nodes to which connections are possible.
- mergecom.srv - Service and sequence definitions.

C.2.4. AE Title/Presentation Address Mapping

Presentation address mapping is configured in the mergecom.app file. The Presentation Address of an SCU/SCP application is specified by configuring the Listen Port in the mergecom.pro file, and specifying the AE title for the SCU/SCP within the application itself.

C.2.5. Configurable Parameters

The mergecom.pro configuration file can be used to set or modify other lower-level communication parameters. This includes time-outs and other parameters. Some information about supported SOP classes is also stored here. **Most parameters in this file should NEVER be changed. Doing so may compromise DICOM conformance.** Before modifying any parameters, such as time-out, be sure to have a backup of the originally supplied mergecom.pro file. Also, before modifying other parameters, you should consider contacting Merge Healthcare for advice.

C.2.6. PDU Size

- The maximum PDU size is configurable with a minimum of 4,096 bytes.
- Application conformance statements specify the chosen PDU (Protocol Data Units) size and any general rules governing the initiation of associations. See the "System Profile" section of the Merge DICOM Reference Manual for further information about configuring the PDU size.

C.3. Extensions/Specializations/Privatizations

C.3.1. Standard Extended/Specialized/Private SOPs

Application conformance statements list extended, specialized, or private SOPs that are supported.

C.3.2. Private Transfer Syntaxes

This section describes private transfer syntaxes that are listed in the Transfer Syntax Tables. See D.3. SYSTEM PROFILE ON PAGE 125 in APPENDIX D. CONFIGURATION PARAMETERS ON PAGE 109 for details

Appendix D. Configuration Parameters

This appendix describes each configuration parameter in detail. Information contained in these tables is the parameter names, descriptions and sections where it is contained. The parameters are listed alphabetically and organized by the initialization file where they are used.

D.1. Initialization File

The following parameters are recognized by Merge DICOM in the initialization file.

Table D.1: Initialization file parameters

Name	Section	Description
BLANK_FILL_LOG_FILE	MergeCOM3	This parameter informs the toolkit whether or not to expand the log file to its maximum size on initialization. Setting this value to "NO" will decrease the time spent in the MC.initialize() call, but increase the time spent doing actual logging while the application is running. DEFAULT: YES
ERROR_MESSAGE	MergeCOM3	This parameter instructs the toolkit to which destination (File, Screen and/or Memory) to log error messages.
INFO_MESSAGE	MergeCOM3	This parameter instructs the toolkit to which destination (File, Screen and/or Memory or none) to log information messages.
LOG_FILE	MergeCOM3	This is the name of the Merge DICOM message log. The file will be [re-]created by Merge DICOM Toolkit. This parameter is ignored by embedded toolkits. The path to the file can be specified using environment variables (including the pseudo environment variable MC3INIDIR which does not need to be set as the toolkit will resolve it internally to the directory where the merge.ini file resides). Unicode paths can now be specified through the environment variables. Both Windows style (%) and Unix style ($) notations for the environment variables are accepted. DEFAULT: ./merge.log
LOG_FILE_BACKUP	MergeCOM3	This is a Boolean parameter that tells Merge DICOM to create a backup of the log file before starting a new log. If "ON", any existing log file is renamed with a file extension of .Lnn where nn is an integer number between 01 and 99. DEFAULT: OFF.

Name	Section	Description
LOG_FILE_LINE_LENGTH	MergeCOM3	This option specifies the number of characters that occur on a line within the merge.log file. DEFAULT: 78 MINIMUM: 16 MAXIMUM: 254
LOG_FILE_SIZE	MergeCOM3	This is the number of lines which will be created for the log file. If BLANK_FILL_LOG_FILE is set to YES, the file is initialized to all binary zeros before the first message is logged. DEFAULT: 1000 MINIMUM: 100 MAXIMUM = 30720 (30 * 1024)
LOG_MEMORY_SIZE	MergeCOM3	This is the number of lines of length equal to LOG_FILE_LINE_LENGTH which will be created for the memory log. Note that this option is ignored when using the .NET Assembly. DEFAULT: 1024.
MERGECOM_3_APPLICATIONS	MergeCOM3	File containing the Merge DICOM application configurations. The path to the file can be specified using environment variables (including the pseudo environment variable MC3INIDIR which does not need to be set as the toolkit will resolve it internally to the directory where the merge.ini file resides). Unicode paths can now be specified through the environment variables. Both Windows style (%) and Unix style ($) notations for the environment variables are accepted.
MERGECOM_3_PROFILE	MergeCOM3	File containing the Merge DICOM system profile parameters. The path to the file can be specified using environment variables (including the pseudo environment variable MC3INIDIR which does not need to be set as the toolkit will resolve it internally to the directory where the merge.ini file resides). Unicode paths can now be specified through the environment variables. Both Windows style (%) and Unix style ($) notations for the environment variables are accepted.

Name	Section	Description
MERGECOM_3_SERVICES	MergeCOM3	File containing the Merge DICOM system service and message definitions. The path to the file can be specified using environment variables (including the pseudo environment variable MC3INIDIR which does not need to be set as the toolkit will resolve it internally to the directory where the merge.ini file resides). Unicode paths can now be specified through the environment variables. Both Windows style (%) and Unix style ($) notations for the environment variables are accepted.
NUM_HISTORICAL_LOG_FILES	MergeCOM3	This parameter informs the toolkit of the number of historical log files to keep. The valid range of number for this parameter is 1 - 99. The historical log files are named basename.L01 to basename.LXX where basename.LXX is the latest log file. The basename is determined by the LOG_FILE parameter. When the maximum number of historical log files is met, the oldest log file is deleted and the log files are renamed. Note that a new log file is created each time the library is initialized. This parameter is only used when LOG_FILE_BACKUP is set to YES.
T1_MESSAGE	MergeCOM3	This logging level is not used (internal tracking).
T2_MESSAGE	MergeCOM3	This logging level parameter instructs the toolkit to log the entire contents of messages sent or received over the network. The format is similar to `MCattributeSet.list`'s output.
T3_MESSAGE	MergeCOM3	This logging level parameter instructs the toolkit to log messages relating to association negotiation.
T4_MESSAGE	MergeCOM3	This logging level parameter instructs the toolkit to log messages when incoming associations are automatically rejected.
T5_MESSAGE	MergeCOM3	This logging level parameter instructs the toolkit to log messages relating to regular and extended validation.
T6_MESSAGE	MergeCOM3	This logging level parameter instructs the toolkit to log messages relating to configuration.
T7_MESSAGE	MergeCOM3	This logging level parameter instructs the toolkit to log messages relating to logging of command level attributes in messages sent or received.
T8_MESSAGE	MergeCOM3	This logging level parameter instructs the toolkit to log messages relating to the streaming in and out of messages and file objects.

Name	Section	Description
T9_MESSAGE	MergeCOM3	This logging level parameter instructs the toolkit to log messages relating to PDU's sent and received. **NOTE:** Receipt and transmission of P-DATA PDU's are logged; not the actual PDU itself.
WARNING_MESSAGE	MergeCOM3	This parameter instructs the toolkit to which destination (File, Screen and/or Memory or none) to log warning messages.

NOTE: The destination for the logging from the T2_MESSAGE to T9_MESSAGE trace levels can be one or more of File, Screen and Memory or none.

D.2. Application Profile

The application profile is a configuration file that is application dependent. The application profile does not set specific parameters. It sets parameters related to characteristics of your own application entity.

This section will define how each parameter should be defined within the application profile.

D.2.1. Sections

The application profile contains the following sections.

Table D.2: Application profile section headings

Section	Description
<remote_application_title>	Section describing a remote DICOM Application Entity title(s). The remote Application Entity titles listed here must be 1 to 16 bytes in length with no embedded spaces. Simply, this section is where you list the DICOM applications you want to communicate with.
<service_list_name>	List(s) of DICOM services that will be provided by the Application Entities listed in the [<remote_application_title>] sections. The service names listed here must be 1 to 33 bytes in length with no embedded spaces. Simply, this section is where you list the services that are provided by the remote DICOM applications.
<syntax_list_name>	List(s) of DICOM transfer syntaxes that will be supported by the services listed in the [<service_list_name>] sections. The transfer syntaxes must be one of those listed in Table D.5, "Transfer Syntax List Parameters," on page 123.

D.2.2. Parameters

The application profile contains the following parameters:

Table D.3: Application profile section headers

Parameter	Section	Description
PORT_NUMBER	<remote_application_title>	This parameter is the TCP/IP port on which the remote DICOM system listens for connections. The commonly used port number is 104. This default value may be overridden by the constructor of the MCremoteApplication class.
HOST_NAME	<remote_application_title>	This parameter is the name of the remote host as it is known to your TCP/IP system. This default value may be overridden by the constructor of the MCremoteApplication class. The parameters value must be 1 to 19 bytes in length with no embedded spaces. NOTE that a numeric internet address may be used: e.g., 192.204.32.1
SERVICE_LIST	<remote_application_title>	This parameter is the name of a section in the application profile which provides a list of services for which local applications will negotiate when attempting to establish an association. This is a default list; another list may be specified in the constructor of the MCremoteApplication class. The parameters value names must be 1 to 33 bytes in length with no embedded spaces.

The SERVICE_LIST section of the Application Profile is used to describe the DICOM services that will be negotiated by the listed Application Entity. The parameter values are text strings recognizable by the Merge DICOM toolkit. These strings are defined in detail in message.txt. This file is located in the mc3msg directory of your distribution. The following is a list of currently supported services:

Table D.4: Application profile parameters

Merge DICOM Toolkit Service Parameter	DICOM Service Class
ACQUISITION_CONTEXT_SR	Storage
ADVANCED_BLENDING_PRESENTATION_STATE	Storage
ARTERIAL_PULSE_WAVEFORM	Storage
AUDIO_WAVEFORM_REAL_TIME_COMMUNICATION	Storage
AUTOREFRACTION_MEASUREMENTS	Storage
BASIC_ANNOTATION_BOX	Print Management
BASIC_COLOR_IMAGE_BOX	Print Management
BASIC_FILM_BOX	Print Management
BASIC_FILM_SESSION	Print Management
BASIC_GRAYSCALE_IMAGE_BOX	Print Management

Merge DICOM Toolkit Service Parameter	DICOM Service Class
BASIC_PRINT_IMAGE_OVERLAY_BOX	Print Management
BASIC_STRUCTURED_DISPLAY	Storage
BODY_POSITION_WAVEFORM	Storage
BREAST_IMAGING_RPI_QUERY	Relevant Patient Information Query
BREAST_PROJ_PRESENT	Storage
BREAST_PROJ_PROCESS	Storage
BREAST_TOMO_IMAGE_STORAGE	Storage
C_ARM_PHOTON_ELECTRON_RADIATION	Storage
C_ARM_PHOTON_ELECTRON_RADIATION_RECORD	Storage
CARDIAC_RPI_QUERY	Relevant Patient Information Query
CHEST_CAD_SR	Storage
COLON_CAD_SR	Storage
COLOR_PALETTE_FIND	Query/Retrieve
COLOR_PALETTE_GET	Query/Retrieve
COLOR_PALETTE_MOVE	Query/Retrieve
COLOR_PALETTE_STORAGE	Storage
COMPOSITE_INST_RET_NO_BULK_GET	Query/Retrieve
COMPOSITE_INSTANCE_ROOT_RET_GET	Query/Retrieve
COMPOSITE_INSTANCE_ROOT_RET_MOVE	Query/Retrieve
COMPOSITING_PLANAR_MPR_VOLUMETRIC_PS	Storage
COMPREHENSIVE_3D_SR	Storage
CONTENT_ASSESSMENT_RESULTS	Storage
CORNEAL_TOPOGRAPHY_MAP	Storage
CT_DEFINED_PROCEDURE_PROTOCOL	Storage
CT_PERFORMED_PROCEDURE_PROTOCOL	Storage
DEFINED_PROCEDURE_PROTOCOL_FIND	Query/Retrieve
DEFINED_PROCEDURE_PROTOCOL_GET	Query/Retrieve
DEFINED_PROCEDURE_PROTOCOL_MOVE	Query/Retrieve
DEFORMABLE_SPATIAL_REGISTRATION	Storage
DERMOSCOPIC_PHOTOGRAPHY_IMAGE	Storage

Merge DICOM Toolkit Service Parameter	DICOM Service Class
DETACHED_INTERP_MANAGEMENT	Results Management
DETACHED_PATIENT_MANAGEMENT	Patient Management
DETACHED_RESULTS_MANAGEMENT	Results Management
DETACHED_STUDY_MANAGEMENT	Study Management
DETACHED_VISIT_MANAGEMENT	Patient Management
DICOMDIR	Media Storage
DISPLAY_SYSTEM	Display System Management
ELECTROMYOGRAM_WAVEFORM	Storage
ELECTROOCULOGRAM_WAVEFORM	Storage
ENCAPSULATED_CDA	Storage
ENCAPSULATED_MTL	Storage
ENCAPSULATED_OBJ	Storage
ENCAPSULATED_STL	Storage
ENHANCED_CONTINUOUS_RT_IMAGE	Storage
ENHANCED_CT_IMAGE	Storage
ENHANCED_MR_COLOR_IMAGE	Storage
ENHANCED_MR_IMAGE	Storage
ENHANCED_PET_IMAGE	Storage
ENHANCED_RT_IMAGE	Storage
ENHANCED_US_VOLUME	Storage
ENHANCED_XA_IMAGE	Storage
ENHANCED_XRAY_RADIATION_DOSE_SR	Storage
ENHANCED_XRF_IMAGE	Storage
EXTENSIBLE_SR	Storage
G_P_PERFORMED_PROCEDURE_STEP_RETIRED	Study Management
G_P_SCHEDULED_PROCEDURE_STEP_RETIRED	Study Management
G_P_WORKLIST_RETIRED	Basic Worklist Management
GENERAL_AUDIO_WAVEFORM	Storage
GENERAL_RPI_QUERY	Relevant Patient Information Query
GENERIC_IMPLANT_TEMPLATE	Storage

Merge DICOM Toolkit Service Parameter	DICOM Service Class
GENERIC_IMPLANT_TEMPLATE_FIND	Query/Retrieve
GENERIC_IMPLANT_TEMPLATE_GET	Query/Retrieve
GENERIC_IMPLANT_TEMPLATE_MOVE	Query/Retrieve
GRAYSCALE_PLANAR_MPR_VOLUMETRIC_PS	Storage
HANGING_PROTOCOL	Hanging Protocol Storage
HANGING_PROTOCOL_FIND	Hanging Protocol Query/Retrieve
HANGING_PROTOCOL_GET	Hanging Protocol Query/Retrieve
HANGING_PROTOCOL_MOVE	Hanging Protocol Query/Retrieve
IMAGE_OVERLAY_BOX_RETIRED	Print Management
IMPLANT_ASSEMBLY_TEMPLATE	Storage
IMPLANT_ASSEMBLY_TEMPLATE_FIND	Query/Retrieve
IMPLANT_ASSEMBLY_TEMPLATE_GET	Query/Retrieve
IMPLANT_ASSEMBLY_TEMPLATE_MOVE	Query/Retrieve
IMPLANT_TEMPLATE_GROUP	Storage
IMPLANT_TEMPLATE_GROUP_FIND	Query/Retrieve
IMPLANT_TEMPLATE_GROUP_GET	Query/Retrieve
IMPLANT_TEMPLATE_GROUP_MOVE	Query/Retrieve
IMPLANTATION_PLAN_SR_DOCUMENT	Storage
INSTANCE_AVAIL_NOTIFICATION	Instance Availability Notification
INTRAOCULAR_LENS_CALCULATIONS	Storage
INVENTORY	Storage
INVENTORY_CREATION	Storage Management
INVENTORY_FIND	Query/Retrieve
INVENTORY_GET	Query/Retrieve
INVENTORY_MOVE	Query/Retrieve
KERATOMETRY_MEASUREMENTS	Storage
KEY_OBJECT_SELECTION_DOC	Storage
LEGACY_CONVERTED_ENHANCED_CT_IMAGE	Storage
LEGACY_CONVERTED_ENHANCED_MR_IMAGE	Storage
LEGACY_CONVERTED_ENHANCED_PET_IMAGE	Storage

Merge DICOM Toolkit Service Parameter	DICOM Service Class
LENSOMETRY_MEASUREMENTS	Storage
MACULAR_GRID_THIICKNESS_VOLUME	Storage
MAMMOGRAPHY_CAD_SR	Storage
MEDIA_CREATION_MANAGEMENT	Media Creation Management
MODALITY_WORKLIST_FIND	Modality Work list
MR_SPECTROSCOPY	Storage
MULTI_CHANNEL_RESPIRATORY_WAVEFORM	Storage
MULTIPLE_VOLUME_RENDERING_VOLUMETRIC_PRESENTATIO N_STATE	Storage
OPHT_VIS_FIELD_STATIC_PERIM_MEAS	Storage
OPHTHALMIC_AXIAL_MEASUREMENTS	Storage
OPHTHALMIC_OCT_BSCAN_VOLUME_ANALYSIS	Storage
OPHTHALMIC_OCT_EN_FACE_IMAGE	Storage
OPHTHALMIC_TOMOGRAPHY_IMAGE	Storage
OPM_THICKNESS_MAP	Storage
PARAMETRIC_MAP	Storage
PATIENT_RADIATION_DOSE_SR	Storage
PATIENT_ROOT_QR_FIND	Query/Retrieve
PATIENT_ROOT_QR_GET	Query/Retrieve
PATIENT_ROOT_QR_MOVE	Query/Retrieve
PATIENT_STUDY_ONLY_QR_FIND_RETIRED	Query/Retrieve
PATIENT_STUDY_ONLY_QR_GET_RETIRED	Query/Retrieve
PATIENT_STUDY_ONLY_QR_MOVE_RETIRED	Query/Retrieve
PERFORMED_IMAGING_AGENT_ADMINISTRATION_SR	Storage
PERFORMED_PROCEDURE_STEP	Study Management
PERFORMED_PROCEDURE_STEP_NOTIFY	Study Management
PERFORMED_PROCEDURE_STEP_RETRIEVE	Study Management
PLANNED_IMAGING_AGENT_ADMINISTRATION_SR	Storage
PRESENTATION_LUT	Print Management
PRINT_JOB	Print Management

Merge DICOM Toolkit Service Parameter	DICOM Service Class
PRINT_QUEUE_MANAGEMENT	Print Management
PRINTER	Print Management
PRINTER_CONFIGURATION	Print Management
PROCEDURAL_EVENT_LOGGING	Application Event Logging
PROCEDURE_LOG	Storage
PRODUCT_CHARACTERISTICS_QUERY	Query/Retrieve
PROTOCOL_APPROVAL	Storage
PROTOCOL_APPROVAL_FIND	Query/Retrieve
PROTOCOL_APPROVAL_MOVE	Query/Retrieve
PROTOCOL_APPROVAL_GET	Query/Retrieve
PULL_PRINT_REQUEST	Print Management
RADIOPHARMACEUTICAL_RADIATION_DOSE_SR	Storage
RAW_DATA	Storage
REAL_WORLD_VALUE_MAPPING	Storage
REFERENCED_IMAGE_BOX	Print Management
RENDITION_SELECTION_DOCUMENT_REAL_TIME_COMMUNICATION	Storage
REPOSITORY_QUERY	Query/Retrieve
RESPIRATORY_WAVEFORM	Storage
ROBOTIC_ARM_RADIATION	Storage
ROBOTIC_ARM_RADIATION_RECORD	Storage
ROUTINE_SCALP_ELECTROENCEPHALOGRAM_WAVEFORM	Storage
RT_BEAMS_DELIVERY_INSTRUCTION	Storage
RT_BRACHY_APP_SETUP_DELIVERY_INSTR	Storage
RT_CONVENTIONAL_MACHINE_VERIFICATION	Verification
RT_ION_MACHINE_VERIFICATION	Verification
RT_PATIENT_POSITION_ACQUISITION_INSTRUCTION	Storage
RT_PHYSICIAN_INTENT	Storage
RT_RADIATION_RECORD_SET	Storage
RT_RADIATION_SALVAGE_RECORD	Storage

Merge DICOM Toolkit Service Parameter	DICOM Service Class
RT_RADIATION_SET	Storage
RT_SEGMENT_ANNOTATION	Storage
SC_MULTIFRAME_GRAYSCALE_BYTE	Storage
SC_MULTIFRAME_GRAYSCALE_WORD	Storage
SC_MULTIFRAME_SINGLE_BIT	Storage
SC_MULTIFRAME_TRUE_COLOR	Storage
SEGMENTATION	Storage
SEGMENTED_VOLUME_RENDERING_VOLUMETRIC_PRESENTATI ON_STATE	Storage
SIMPLIFIED_ADULT_ECHO_SR	Storage
SLEEP_ELECTROENCEPHALOGRAM_WAVEFORM	Storage
SPATIAL_FIDUCIALS	Storage
SPATIAL_REGISTRATION	Storage
SPECTACLE_PRESCRIPTION_REPORT	Storage
STANDARD_BASIC_TEXT_SR	Storage
STANDARD_BLENDING_SOFTCOPY_PS	Storage
STANDARD_COLOR_SOFTCOPY_PS	Storage
STANDARD_COMPREHENSIVE_SR	Storage
STANDARD_CR	Storage
STANDARD_CT	Storage
STANDARD_CURVE	Storage
STANDARD_DX_PRESENT	Storage
STANDARD_DX_PROCESS	Storage
STANDARD_ECHO	Verification
STANDARD_ENCAPSULATED_PDF	Storage
STANDARD_ENHANCED_SR	Storage
STANDARD_GRAYSCALE_SOFTCOPY_PS	Storage
STANDARD_HARDCOPY_COLOR	Storage
STANDARD_HARDCOPY_GRAYSCALE	Storage
STANDARD_IO_PRESENT	Storage

Merge DICOM Toolkit Service Parameter	DICOM Service Class
STANDARD_IO_PROCESS	Storage
STANDARD_IVOCT_PRESENT	Storage
STANDARD_IVOCT_PROCESS	Storage
STANDARD_MG_PRESENT	Storage
STANDARD_MG_PROCESS	Storage
STANDARD_MODALITY_LUT	Storage
STANDARD_MR	Storage
STANDARD_NM	Storage
STANDARD_NM_RETIRED	Storage
STANDARD_OPHTHALMIC_16_BIT	Storage
STANDARD_OPHTHALMIC_8_BIT	Storage
STANDARD_OVERLAY	Storage
STANDARD_PET	Storage
STANDARD_PET_CURVE	Storage
STANDARD_PRINT_STORAGE	Storage
STANDARD_PSEUDOCOLOR_SOFTCOPY_PS	Storage
STANDARD_RT_BEAMS_TREAT	Storage
STANDARD_RT_BRACHY_TREAT	Storage
STANDARD_RT_DOSE	Storage
STANDARD_RT_IMAGE	Storage
STANDARD_RT_ION_BEAMS_TREAT	Storage
STANDARD_RT_ION_PLAN	Storage
STANDARD_RT_PLAN	Storage
STANDARD_RT_STRUCTURE_SET	Storage
STANDARD_RT_TREAT_SUM	Storage
STANDARD_SEC_CAPTURE	Storage
STANDARD_US	Storage
STANDARD_US_MF	Storage
STANDARD_US_MF_RETIRED	Storage
STANDARD_US_RETIRED	Storage

Merge DICOM Toolkit Service Parameter	DICOM Service Class
STANDARD_VIDEO_ENDOSCOPIC	Storage
STANDARD_VIDEO_MICROSCOPIC	Storage
STANDARD_VIDEO_PHOTOGRAPHIC	Storage
STANDARD_VL_ENDOSCOPIC	Storage
STANDARD_VL_MICROSCOPIC	Storage
STANDARD_VL_PHOTOGRAPHIC	Storage
STANDARD_VL_SLIDE_MICROSCOPIC	Storage
STANDARD_VOI_LUT	Storage
STANDARD_WAVEFORM_12_LEAD_ECG	Storage
STANDARD_WAVEFORM_AMBULATORY_ECG	Storage
STANDARD_WAVEFORM_BASIC_VOICE_AUDIO	Storage
STANDARD_WAVEFORM_CARDIAC_EP	Storage
STANDARD_WAVEFORM_GENERAL_ECG	Storage
STANDARD_WAVEFORM_HEMODYNAMIC	Storage
STANDARD_XRAY_ANGIO	Storage
STANDARD_XRAY_ANGIO_BIPLANE	Storage
STANDARD_XRAY_RF	Storage
STEREOMETRIC_RELATIONSHIP	Storage
STORAGE_COMMITMENT_PULL	Storage Commitment
STORAGE_COMMITMENT_PUSH	Storage Commitment
STUDY_COMPONENT_MANAGEMENT	Study Management
STUDY_CONTENT_NOTIFICATION	Study Content Notification
STUDY_ROOT_QR_FIND	Query/Retrieve
STUDY_ROOT_QR_GET	Query/Retrieve
STUDY_ROOT_QR_MOVE	Query/Retrieve
SUBJ_REFRACTION_MEASUREMENTS	Storage
SUBSTANCE_ADMIN_LOGGING	Storage
SUBSTANCE_APPROVAL_QUERY	Storage
SURFACE_SCAN_MESH	Storage
SURFACE_SCAN_POINT_CLOUD	Storage

Merge DICOM Toolkit Service Parameter	DICOM Service Class
SURFACE_SEGMENTATION	Storage
TOMOTHERAPEUTIC_RADIATION	Storage
TOMOTHERAPEUTIC_RADIATION_RECORD	Storage
TRACTOGRAPHY_RESULTS	Storage
UPS_EVENT_SOP	Unified Procedure Step Management
UPS_EVENT_SOP_TRIAL_RETIRED	Unified Procedure Step Management
UPS_PULL_SOP	Unified Procedure Step Management
UPS_PULL_SOP_TRIAL_RETIRED	Unified Procedure Step Management
UPS_PUSH_SOP	Unified Procedure Step Management
UPS_PUSH_SOP_TRIAL_RETIRED	Unified Procedure Step Management
UPS_QUERY_SOP	Unified Procedure Step Management
UPS_WATCH_SOP	Unified Procedure Step Management
UPS_WATCH_SOP_TRIAL_RETIRED	Unified Procedure Step Management
VARIABLE_MODALITY_LUT_SOFTCOPY_PRESENTATION_STATE	Storage
VIDEO_ENDOSCOPIC_IMAGE_REAL_TIME_COMMUNICATION	Storage
VIDEO_PHOTOGRAPHIC_IMAGE_REAL_TIME_COMMUNICATIO N	Storage
VISUAL_ACUITY_MEASUREMENTS	Storage
VL_WHOLE_SLIDE_MICROSCOPY_IMAGE	Storage
VOI_LUT_BOX	Print Management
VOLUME_RENDERING_VOLUMETRIC_PRESENTATION_STATE	Storage
WIDE_FIELD_OPHTHALMIC_PHOTO_3D_COORDINATES	Storage
WIDE_FIELD_OPHTHALMIC_PHOTO_STEREOGRAPHIC_PROJ	Storage
XA_DEFINED_PROCEDURE_PROTOCOL	Storage
XA_PERFORMED_PROCEDURE_PROTOCOL	Storage
XA_XRF_GRAYSCALE_SOFTCOPY_PS	Storage
XRAY_3D_ANGIO_IMAGE	Storage
XRAY_3D_CRANIO_IMAGE	Storage
XRAY_RADIATION_DOSE_SR	Storage
BASIC_COLOR_PRINT_MANAGEMENT (META_SOP)	Print Management

Merge DICOM Toolkit Service Parameter	DICOM Service Class
BASIC_GRAYSCALE_PRINT_MANAGEMENT (META_SOP)	Print Management
DETACHED_PATIENT_MANAGEMENT_META (META_SOP)	Print Management
DETACHED_RESULTS_MANAGEMENT_META (META_SOP)	Results Management
G_P_WORKLIST_MANAGEMENT_META _RETIRED (META_SOP)	Basic Worklist Management
PULL_STORED_PRINT_MANAGEMENT (META_SOP)	Print Management
REF_COLOR_PRINT_MANAGEMENT (META_SOP)	Print Management
REF_GRAYSCALE_PRINT_MANAGEMENT (META_SOP)	Print Management
STUDY_MANAGEMENT (META_SOP)	Study Management

Transfer syntax lists are contained in the service lists. The following is a list of the currently supported transfer syntaxes.

Table D.5: Transfer Syntax List Parameters

Merge DICOM Transfer Syntax Parameter	Description
IMPLICIT_LITTLE_ENDIAN	Implicit VR Little Endian: Default Transfer Syntax for DICOM
IMPLICIT_BIG_ENDIAN	Implicit VR Big Endian
ENCAPSULATED_UNCOMPRESSED_ELE	Encapsulated Uncompressed Explicit VR Little Endian
EXPLICIT_LITTLE_ENDIAN	Explicit VR Little Endian
EXPLICIT_BIG_ENDIAN	Explicit VR Big Endian
RLE	Run length Encoding
DEFLATED_EXPLICIT_LITTLE_ENDIAN	Deflated Explicit VR Little Endian
JPEG_BASELINE	JPEG Baseline (Process 1): Default Transfer Syntax for Lossy JPEG 8 Bit Image Compression
JPEG_EXTENDED_2_4	JPEG Extended (Process 2 & 4): Default Transfer Syntax for Lossy JPEG 12 Bit Image Compression (Process 4 only)
JPEG_EXTENDED_3_5	JPEG Extended (Process 3 & 5)
JPEG_SPEC_NON_HIER_6_8	JPEG Spectral Selection, Non-Hierarchical (Process 6 & 8)
JPEG_SPEC_NON_HIER_7_9	JPEG Spectral Selection, Non-Hierarchical (Process 7 & 9)
JPEG_FULL_PROG_NON_HIER_10_12	JPEG Full Progression, Non-Hierarchical (Process 10 & 12)

Merge DICOM Transfer Syntax Parameter	Description
JPEG_FULL_PROG_NON_HIER_11_13	JPEG Full Progression, Non-Hierarchical (Process 11 & 13)
JPEG_LOSSLESS_NON_HIER_14	JPEG Lossless, Non-Hierarchical (Process 14)
JPEG_LOSSLESS_NON_HIER_15	JPEG Lossless, Non-Hierarchical (Process 15)
JPEG_EXTENDED_HIER_16_18	JPEG Extended, Hierarchical (Process 16 & 18)
JPEG_EXTENDED_HIER_17_19	JPEG Extended, Hierarchical (Process 17 & 19)
JPEG_SPEC_HIER_20_22	JPEG Spectral Selection, Hierarchical (Process 20 & 22)
JPEG_SPEC_HIER_21_23	JPEG Spectral Selection, Hierarchical (Process 21 & 23)
JPEG_FULL_PROG_HIER_24_26	JPEG Full Progression, Hierarchical (Process 24 & 26)
JPEG_FULL_PROG_HIER_25_27	JPEG Full Progression, Hierarchical (Process 25 & 27)
JPEG_LOSSLESS_HIER_28	JPEG Lossless, Hierarchical (Process 28)
JPEG_LOSSLESS_HIER_29	JPEG Lossless, Hierarchical (Process 29)
JPEG_LOSSLESS_HIER_14	JPEG Lossless, Hierarchical, First-Order Prediction (Process 14 [Selection Value 1]): Default Transfer Syntax for Lossless JPEG Image Compression
JPEG_2000_LOSSLESS_ONLY	JPEG 2000, Lossless
JPEG_2000	JPEG 2000, Lossless or Lossy
JPEG_LS_LOSSLESS	JPEG LS Lossless
JPEG_LS_LOSSY	JPEG LS Lossy (Near-Lossless)
JPEG_2000_MC_LOSSLESS_ONLY	JPEG 2000 Part 2 Multi-component Image Compression (Lossless Only)
JPEG_2000_MC	JPEG 2000 Part 2 Multi-component Image Compression
HEVC_H265_M10P_LEVEL_5_1	HEVC/H.265 Main 10 Profile / Level 5.1
HEVC_H265_MP_LEVEL_5_1	HEVC/H.265 Main Profile / Level 5.1
JPIP_REFERENCED	JPIP Referenced
JPIP_REFERENCED_DEFLATE	JPIP Referenced Deflate

Merge DICOM Transfer Syntax Parameter	Description
MPEG2_MPHL	MPEG2 Main Profile @ High Level
MPEG2_MPML	MPEG2 Main Profile @ Main Level
MPEG4_AVC_H264_HP_LEVEL_4_1	MPEG-4 AVC/H.264 High Profile / Level 4.1
MPEG4_AVC_H264_BDC_HP_LEVEL_4_1	MPEG-4 AVC/H.264 BDcompatible High Profile / Level 4.1
MPEG4_AVC_H264_HP_LEVEL_4_2_2D	MPEG-4 AVC/H.264 High Profile / Level 4.2 For 2D Video
MPEG4_AVC_H264_HP_LEVEL_4_2_3D	MPEG-4 AVC/H.264 High Profile / Level 4.2 For 3D Video
MPEG4_AVC_H264_STEREO_HP_LEVEL_4_2	MPEG-4 AVC/H.264 Stereo High Profile / Level 4.2
SMPTE_ST_2110_20_UNCOMPRESSED_PROGRESSIVE_ACTIVE_VIDEO	SMPTE ST 2110-20 Uncompressed Progressive Active Video
SMPTE_ST_2110_20_UNCOMPRESSED_INTERLACED_ACTIVE_VIDEO	SMPTE ST 2110-20 Uncompressed Interlaced Active Video
SMPTE_ST_2110_30_PCM_DIGITAL_AUDIO	SMPTE ST 2110-30 PCM Digital Audio
PRIVATE_SYNTAX_1	Private transfer syntax 1 with the characteristics specified by the PRIVATE_SYNTAX_1_LITTLE_ENDIAN, PRIVATE_SYNTAX_1_EXPLICIT_VR, and PRIVATE_SYNTAX_1_ENCAPSULATED configuration options.
PRIVATE_SYNTAX_2	Private transfer syntax 2 with the characteristics specified by the PRIVATE_SYNTAX_2_LITTLE_ENDIAN, PRIVATE_SYNTAX_2_EXPLICIT_VR, and PRIVATE_SYNTAX_2_ENCAPSULATED configuration options.

D.3. System Profile

The System Profile is used to define system-wide parameters. These parameters apply across all associations with other DICOM application entities. The location of this file is provided by the `MERGECOM_3_PROFILE` parameter of the `[MergeCOM3]` section of the `MERGE.INI` file.

The following are a few notes to keep in mind concerning the System Profile:

You must specify your own unique DICOM Implementation Class UID and place it in this file along with an optional Implementation Version. These need to be documented in your DICOM conformance statement.

There are several exception options specified at both the association and DIMSE levels of DICOM communication. You should not have to modify these options in normal circumstances and doing so could make your application non DICOM conformant.

The DICOM Upper Layer section network time-outs can be modified. This is useful on slower or less- predictable networks (e.g., WAN's).

The section of the System Profile dealing with transport parameters is important. This is where you specify the **TCP/IP listen port** for a DICOM server (SCP) application, along with the **number of simultaneous associations** your server will support over this port.

Table D.6, "[ASSOC_PARMS] section of system profile parameters," on page 126 through Table D.11, "[TRANSPORT_PARMS] section of system profile parameters," on page 146 define how each parameter should be defined within the system profile.

Table D.6: [ASSOC_PARMS] section of system profile parameters

Name	Description
ACCEPT_ANY_APPLICATION_TITLE †	If set to YES, the remote system need not specify a correct DICOM application title when requesting an association. If set to NO a correct application title must be used. When this value is set to YES, the toolkit will report the remote application as connecting to the first application registered. DEFAULT: NO
ACCEPT_ANY_CONTEXT_NAME †	If set to YES, the remote system need not specify the LOCAL_APPL_CONTEXT_NAME when requesting an association. If set to NO, the correct context name must be used. DEFAULT: NO
ACCEPT_ANY_HOSTNAME	If set to YES, the toolkit will not check if applications connecting to an SCP can have their hostname resolved through the SCP's host file or domain name server. If set to NO, the toolkit will automatically reject associations from unknown hosts. DEFAULT: NO
ACCEPT_ANY_PRESENTATION_CONTEXT †	If set to YES, the toolkit will not validate that the presentation context ID contained in a message's PDU header information matches the ID of the presentation context negotiated for the type of message contained in the PDU. If set to NO, the toolkit will abort associations when these values do not match. DEFAULT: NO
ACCEPT_DIFFERENT_IC_UID †	If set to NO, the remote system must specify the local IMPLEMENTATION_CLASS_UID when requesting an association. If set to YES, a different implementation class UID may be used. DEFAULT: YES

Name	Description
ACCEPT_DIFFERENT_VERSION †	If set to NO, the remote system must specify the local IMPLEMENTATION_VERSION when requesting an association. If set to YES, a different implementation version may be used. DEFAULT: YES
ACCEPT_LIST_OF_APPLICATION_TITLES	List of AE titles which the remote system might use when requesting an association. The parameters line should contain all AE titles separated by one of predefined delimiters: ',' '\' '/' ';'. The length of each AE title cannot exceed 16 characters. Example: ACCEPT_LIST_OF_APPLICATION_TITLES = MERGE_STORE_SCP/MERGE_STORE_SCU/ MERGE_STORE_RQ DEFAULT: <none>
ACCEPT_MULTIPLE_PRES_CONTEXTS	If set to YES, SCP applications will allow multiple presentation contexts to be negotiated for a single DICOM service. If set to NO, an SCP will only accept a single presentation context for a DICOM service. DEFAULT:YES
ACCEPT_RELATED_GENERAL_SERVICES	This parameter sets the Merge DICOM Toolkit behavior in regard to support for DICOM Supplement 90. Supplement 90 defines a method for association requestors to specify the generalized version of a SOP Class. When set to YES, Merge DICOM Toolkit will allow association acceptors to accept a presentation context whose generalized SOP Class is supported; however, the customized SOP Class is not specifically supported. DEFAULT: NO
ACCEPT_STORAGE_SERVICE_CONTEXTS	This parameter sets the Merge DICOM Toolkit behavior in regard to support for DICOM Supplement 90. When set to YES, Merge DICOM Toolkit will accept any presentation context which is defined as a Storage Service Class SOP Class. DEFAULT: NO
ALLOW_EMPTY_PDV_LENGTH	The DICOM standard specifies that PDVs shall not be sent without any content in the fragment. The toolkit however can send and accept empty PDVs. To enforce the standard requirement, this setting should be set to No. DEFAULT: YES

Name	Description
AUTO_ECHO_SUPPORT	If set to YES, the toolkit automatically handles C-ECHO requests when the application doesn't explicitly include STANDARD_ECHO in its supported service list. If set to NO, the toolkit rejects C-ECHO requests when the application doesn't explicitly include STANDARD_ECHO in its supported service list. DEFAULT: YES
DEFLATED_EXPLICIT_LITTLE_ENDIAN_SYNTAX	This value defines the UID of the Deflated explicit VR little endian transfer syntax. DEFAULT: 1.2.840.10008.1.2.1.99
ENCAPSULATED_UNCOMPRESSED_ELE_SYNTAX	This value defines the UID of the Encapsulated Uncompressed Explicit VR Little Endian transfer syntax. DEFAULT: 1.2.840.10008.1.2.1.98
EXPLICIT_BIG_ENDIAN_SYNTAX	This value defines the UID of the Explicit VR Big Endian transfer syntax. This transfer syntax has been retired by the DICOM standard. DEFAULT: 1.2.840.10008.1.2.2
EXPLICIT_LITTLE_ENDIAN_SYNTAX	This value defines the UID of the Explicit VR Little Endian transfer syntax. DEFAULT: 1.2.840.10008.1.2.1
HARD_CLOSE_TCP_IP_CONNECTION	This parameter specifies how TCP/IP connections are closed by the toolkit. When set to YES, TCP/IP connections are instantaneously closed with an RST packet. When set to NO, TCP/IP connections are closed gracefully with a FIN packet. Note, that in the NO case the toolkit must wait for an operating system dependent amount of time for the response to the FIN packet. DEFAULT: YES
IMPLEMENTATION_CLASS_UID	The DICOM Implementation Class UID (as specified in your DICOM conformance statement).
IMPLEMENTATION_VERSION	The Implementation Version Number (as specified in your DICOM conformance statement).
IMPLICIT_BIG_ENDIAN_SYNTAX	The Implicit VR Big Endian transfer syntax is not defined by the DICOM standard. This value is provided to supply compatibility with private implementations. DEFAULT: <none>
IMPLICIT_LITTLE_ENDIAN_SYNTAX	The Implicit VR Little Endian transfer syntax is the default network transfer syntax of the DICOM standard. The Implicit VR Little Endian transfer syntax must always be defined. DEFAULT: 1.2.840.10008.1.2

Name	Description
JPEG_2000_LOSSLESS_ONLY_SYNTAX	This value defines the UID for JPEG 2000, Lossless transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.90
JPEG_2000_MC_LOSSLESS_ONLY_SYNTAX	This value defines the UID for JPEG 2000 Part 2 Multi- component Image Compression (Lossless Only) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.92
JPEG_2000_MC_SYNTAX	This value defines the UID for JPEG 2000 Part 2 Multi- component Image Compression transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.93
JPEG_2000_SYNTAX	This value defines the UID for JPEG 2000 transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.91
JPEG_BASELINE_SYNTAX	This value defines the UID for JPEG Baseline (Process 1) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.50
JPEG_EXTENDED_2_4_SYNTAX	This value defines the UID for JPEG Extended (Process 2 & 4) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.51
JPEG_EXTENDED_3_5_SYNTAX	This value defines the UID for JPEG Extended (Process 3 & 5) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.52
JPEG_EXTENDED_HIER_16_18_SYNTAX	This value defines the UID for JPEG Extended, Hierarchical (Process 16 & 18) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.59
JPEG_EXTENDED_HIER_17_19_SYNTAX	This value defines the UID for JPEG Extended, Hierarchical (Process 17 & 19) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.60
JPEG_FULL_PROG_HIER_24_26_SYNTAX	This value defines the UID for JPEG Full Progression, Hierarchical (Process 24 & 26) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.63
JPEG_FULL_PROG_HIER_25_27_SYNTAX	This value defines the UID for JPEG Full Progression, Hierarchical (Process 25 & 27) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.64
JPEG_FULL_PROG_NON_HIER_10_12_SYNTAX	This value defines the UID for JPEG Full Progression, Non-Hierarchical (Process 10 & 12) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.55

Name	Description
JPEG_FULL_PROG_NON_HIER_11_13_SYNTAX	This value defines the UID for JPEG Full Progression, Non-Hierarchical (Process 11 & 13) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.56
JPEG_LOSSLESS_HIER_14_SYNTAX	This value defines the UID for JPEG Lossless, Non-Hierarchical, First-Order Prediction (Process 14, Selection Value 1) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.70
JPEG_LOSSLESS_HIER_28_SYNTAX	This value defines the UID for JPEG Lossless, Hierarchical (Process 28) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.65
JPEG_LOSSLESS_HIER_29_SYNTAX	This value defines the UID for JPEG Lossless, Hierarchical (Process 29) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.66
JPEG_LOSSLESS_NON_HIER_14_SYNTAX	This value defines the UID for JPEG Lossless, Non-Hierarchical (Process 14) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.57
JPEG_LOSSLESS_NON_HIER_15_SYNTAX	This value defines the UID for JPEG Lossless, Non-Hierarchical (Process 15) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.58
JPEG_LS_LOSSLESS_SYNTAX	This value defines the UID for JPEG LS Lossless transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.80
JPEG_LS_LOSSY_SYNTAX	This value defines the UID for JPEG LS Lossy (Near Lossless) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.81
JPEG_SPEC_HIER_20_22_SYNTAX	This value defines the UID for JPEG Spectral Selection, Hierarchical (Process 20 & 22) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.61
JPEG_SPEC_HIER_21_23_SYNTAX	This value defines the UID for JPEG Spectral Selection, Hierarchical (Process 21 & 23) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.62
JPEG_SPEC_NON_HIER_6_8_SYNTAX	This value defines the UID for JPEG Spectral Selection, Non Hierarchical (Process 6 & 8) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.53
JPEG_SPEC_NON_HIER_7_9_SYNTAX	This value defines the UID for JPEG Spectral Selection, Non Hierarchical (Process 7 & 9) transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.54

Name	Description
JPIP_REFERENCED_DEFLATE_SYNTAX	This value defines the UID for JPIP Referenced Deflate transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.95
JPIP_REFERENCED_SYNTAX	This value defines the UID for JPIP Referenced transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.94
LICENSE	The Merge DICOM Toolkit license number that was supplied when the toolkit was purchased.
LOCAL_APPL_CONTEXT_NAME	The DICOM Application Context Name (UID) (as specified in the DICOM Standard). DEFAULT: 1.2.840.10008.3.1.1.1
MPEG2_MPHL_SYNTAX	This value defines the UID for MPEG2 Main Profile @ High Level transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.101
MPEG2_MPML_SYNTAX	This value defines the UID for MPEG2 Main Profile @ Main Level transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.100
MPEG4_AVC_H264_BDC_HP_LEVEL_4_1_SYNT AX	This value defines the UID for MPEG-4 AVC/H.264 BD compatible High Profile / Level 4.1 transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.103
MPEG4_AVC_H264_HP_LEVEL_4_1_SYNTAX	This value defines the UID for MPEG-4 AVC/H.264 High Profile / Level 4.1 transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.102
MPEG4_AVC_H264_HP_LEVEL_4_2_2D_SYNTA X	This value defines the UID for MPEG-4 AVC/H.264 High Profile / Level 4.2For 2D Video transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.104
MPEG4_AVC_H264_HP_LEVEL_4_2_3D_SYNTA X	This value defines the UID for MPEG-4 AVC/H.264 High Profile / Level 4.2For 3D Video transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.105
MPEG4_AVC_H264_STEREO_HP_LEVEL_4_2_S YNTAX	This value defines the UID for MPEG-4 AVC/H.264 Stereo High Profile /Level 4.2 transfer syntax. DEFAULT: 1.2.840.10008.1.2.4.106

Name	Description
PDU_MAXIMUM_LENGTH *	The maximum size of Protocol Data Units that can be received by this Merge DICOM Toolkit implementation. This value will also place a limit on how large PDU values being sent can be. Setting this so that a PDU fits within an even multiple of the default TCP/IP MSS (Maximum Segment Size) of 1460 will optimize network performance. Note that 6 bytes for the PDU header must be added to the configured maximum PDU size when calculating a multiple of the MSS. Note also to see the TCPIP_SEND_BUFFER_SIZE and TCPIP_RECEIVE_BUFFER_SIZE configuration values for improving performance. Example: (1460*44)-6 = 64234 PDU Size DEFAULT: 64234 MINIMUM: 4K MAXIMUM: NONE
PRIVATE_SYNTAX_1_ENCAPSULATED	When set to YES, Merge DICOM Toolkit will interpret private transfer syntax 1 as having its pixel data tag (7fe0,0010) being encoded as undefined length in the same manner as the JPEG and RLE transfer syntaxes are encoded. DEFAULT: NO
PRIVATE_SYNTAX_1_EXPLICIT_VR	When set to YES, Merge DICOM Toolkit will interpret private transfer syntax 1 as being encoded in explicit VR format. DEFAULT: YES
PRIVATE_SYNTAX_1_LITTLE_ENDIAN	When set to YES, Merge DICOM Toolkit will interpret private transfer syntax 1 as being encoded in little endian format. DEFAULT: YES
PRIVATE_SYNTAX_1_SYNTAX	The unique identifier (UID) Merge DICOM Toolkit will use to identify private transfer syntax 1. When this value is set to "<none>", private transfer syntax support is shut off. DEFAULT: <none>
PRIVATE_SYNTAX_2_ENCAPSULATED	When set to YES, Merge DICOM Toolkit will interpret private transfer syntax 2 as having its pixel data tag (7fe0,0010) being encoded as undefined length in the same manner as the JPEG and RLE transfer syntaxes are encoded. DEFAULT: NO
PRIVATE_SYNTAX_2_EXPLICIT_VR	When set to YES, Merge DICOM Toolkit will interpret private transfer syntax 2 as being encoded in explicit VR format. DEFAULT: YES

Name	Description
PRIVATE_SYNTAX_2_LITTLE_ENDIAN	When set to YES, Merge DICOM Toolkit will interpret private transfer syntax 2 as being encoded in little endian format. DEFAULT: YES
PRIVATE_SYNTAX_2_SYNTAX	The unique identifier (UID) Merge DICOM Toolkit will use to identify private transfer syntax 2. When this value is set to "<none>", private transfer syntax support is shut off. DEFAULT: <none>
RLE_SYNTAX	This value defines the UID of the RLE Lossless transfer syntax. DEFAULT: 1.2.840.10008.1.2.5
SMPTE_ST_2110_20_UNCOMPRESSED_INTERLACED_ACTIVE_VIDEO_SYNTAX	This value defines the UID for SMPTE ST 2110-20 Uncompressed Interlaced Active Video transfer syntax. DEFAULT: 1.2.840.10008.1.2.7.2
SMPTE_ST_2110_20_UNCOMPRESSED_PROGRESSIVE_ACTIVE_VIDEO_SYNTAX	This value defines the UID for SMPTE ST 2110-20 Uncompressed Progressive Active Video transfer syntax. DEFAULT: 1.2.840.10008.1.2.7.1
SMPTE_ST_2110_30_PCM_DIGITAL_AUDIO_SYNTAX	This value defines the UID for SMPTE ST 2110-30 PCM Digital Audio transfer syntax. DEFAULT: 1.2.840.10008.1.2.7.3

† **These options allow for non-standard DICOM operations.** Such exceptions, if used, should be noted in your DICOM conformance statement.

* Performance tuning.

Table D.7: [DIMSE_PARMS] section of system profile parameters

Name	Description
INITIATOR_NAME †	The DICOM standard has retired the old ACR/NEMA Initiator Name attribute in command messages. To generate such an attribute in command messages, specify an initiator name. <none> means do not put initiator name in messages. DEFAULT: <none>
RECEIVER_NAME †	The DICOM standard has retired the old ACR/NEMA Receiver Name attribute in command messages. To generate such an attribute in command messages, specify a receiver name. <none> means do not put receiver name in messages. DEFAULT: <none>

Name	Description
SEND_ECHO_PRIORITY †	The DICOM standard has retired the message priority attribute in echo command messages. To generate such an attribute in command messages, specify YES. To NOT use message priority in echo messages, specify NO. DEFAULT: NO
SEND_LENGTH_TO_END †	The DICOM standard has retired the old Group-Length-To-End attribute in command messages. To generate such an attribute in command messages, specify YES. If you do not want to generate Group Length To End, specify NO. DEFAULT: NO
SEND_MSG_ID_RESPONSE †	The DICOM standard has retired the message ID attribute in response command messages. To generate such an attribute in command messages, specify YES. To NOT use message ID in response mes- sages, specify NO. DEFAULT: NO
SEND_RECOGNITION_CODE †	The DICOM standard has retired the old Recognition Code attribute in command messages. To generate such an attribute in command messages, specify YES. If you do not want to generate such an attribute, specify NO. DEFAULT: NO
SEND_RESPONSE_PRIORITY †	The DICOM standard has retired the message priority attribute in re- sponse messages. To generate such an attribute in response messages, specify YES. To NOT use message priority in response messages, specify NO. DEFAULT: NO
SEND_SOP_CLASS_UID †	Certain DICOM service classes demand that the affected SOP class UID be present in the message. To prevent the library from ensuring that this is done, specify NO. To ensure that Affected SOP class UID is present, specify YES. DEFAULT: YES
SEND_SOP_INSTANCE_UID †	Certain DICOM service classes demand that the affected SOP instance UID be present in the message. To prevent the library from ensuring that this is done, specify NO. To ensure that Affected SOP instance UID is present, specify YES. DEFAULT: YES

† **These options allow for non-standard DICOM operations.** Such exceptions, if used, should be noted in your DICOM conformance statement.

Table D.8: [DUL_PARMS] section of system profile parameters

Name	Description
ARTIM_TIMEOUT	The number of seconds to use as a time out waiting for an association request or waiting for the peer to shut down an association. DEFAULT: 30
ASSOC_REPLY_TIMEOUT	The number of seconds to wait for a reply to an associate request. DEFAULT: 15.

Name	Description
CONNECT_TIMEOUT	The number of seconds to wait for a network connect to be accepted. DEFAULT: 15.
INACTIVITY_TIMEOUT	The number of seconds to wait in between packets of data received over the network after the initial packet of data in a message is received. Used by the `MCassociation.read()`, `MCassociation.continueRead()`, `MCassociation.readToStream()` and `MCassociation.continueReadToStream()` methods. DEFAULT: 15.
INSURE_EVEN_UID_LENGTH †	Set to NO, if odd-length UIDs in PDU's should NOT be padded with a NULL to ensure even length unique Ids. Set to YES to ensure even UIDs in PDUs. DEFAULT: NO
RELEASE_TIMEOUT	The number of seconds to wait for a reply to an associate release. DEFAULT: 15.
WRITE_TIMEOUT	The number of seconds to wait for a network write to be accepted. DEFAULT: 15.

† These options allow for non-standard DICOM operations. Such exceptions, if used, should be noted in your DICOM conformance statement.

Table D.9: [MEDIA_PARMS] section of system profile parameters

Name	Description
DICOMDIR_STREAM_STORAGE	When set to yes, DICOMDIRs read in leave their directory records in- ternally in "stream" format and are not parsed until the directory record is referenced. This can greatly reduce memory usage when reading in large DICOMDIRs when the entire DICOMDIR is not referenced. Default: NO
EXPORT_GROUP_LENGTHS_TO_M EDIA*	When set to NO, do not write group length attributes with `MCfile.writeP10File()`. DEFAULT: YES
EXPORT_PRIVATE_ATTRIBUTES_TO_MEDIA	When set to NO, disable the exporting of private attributes in files writ- ten with the `MCfile.writeP10File()`. DEFAULT: YES
EXPORT_UN_VR_TO_MEDIA	When set to NO, disable the exporting of attributes with a VR of UN in files written with the `MCfile.writeP10File()`. DEFAULT: YES
EXPORT_UNDEFINED_LENGTH_S Q_IN_DICOMDIR*	When set to NO, DICOMDIRs written with `MCfile.writeP10File()` are created with their sequence attributes having defined lengths. Setting this option to Yes will increase performance. DEFAULT: YES

* Performance tuning.

Table D.10: {MESSAGE_PARMS] section of system profile parameters

Name	Description
ALLOW_COMMA_IN_DS_FL_FD_STRINGS	When set to Yes, a comma or a period will be allowed in the string value passed to MCattributeSet.addValue() for attributes with a VR of DS, FL or FD. When set to No, only a period will be acceptable as a decimal separator. Note that the toolkit will always ensure that DS attributes use a period decimal separator when streaming to the network or to a file, regardless of current locale settings. DEFAULT: NO
ALLOW_INVALID_LENGTH_FOR_VR	When set to 'Yes', data values of fixed length value representations (SS, US, AT, SL, UL, SV, UV, FL, FD) with incorrect length, according to DICOM, will not cause an MC_INVALID_LENGTH_FOR_VR error on DICOM data reading. DEFAULT: YES
ALLOW_INVALID_PRIVATE_ATTRIBUTES	When reading messages or file objects, this parameter specifies if private attributes encoded in an invalid format should be ignored or parsed. DEFAULT: NO
ALLOW_INVALID_PRIVATE_CREATOR_CODES	When reading messages or file objects, this parameter specifies if private creator codes encoded with invalid characters should be ignored or parsed. DEFAULT: NO
ALLOW_OUT_OF_RANGE_BITS_JPEG_LOSSLESS	During decompression of JPEG lossless images, the Pegasus decompressor may discover that the original compressor had failed to mask off the out-of-range bits for the image bit depth. However, if all other lossless JPEG computations are correct, the original image, including such incorrect out-of-range bits, can be losslessly recovered. The Pegasus decompressor will return a warning status, along with the fully decoded image. If this flag is set by the application, the out-of-range bits in output pixels will not be masked off, but returned in the decoded image. Without this flag, out- of-range bits will be masked off to keep pixel values in range. DEFAULT: NO
ATT_00081190_USE_UT_VR	In the 2014b edition of the DICOM Standard, the value representation of attribute (0008,1190) Retrieve URL was changed from UT to the newly introduced UR. For backward compatibility, this parameter specifies that, when reading messages or file objects, the attribute is expected to have the old UT value representation. DEFAULT: NO

Name	Description
ATT_00287FE0_USE_UT_VR	In the 2014b edition of the DICOM Standard, the value representation of attribute (0028,7FE0) Pixel Data Provider URL was changed from UT to the newly introduced UR. For backward compatibility, this parameter specifies that, when reading messages or file objects, the attribute is expected to have the old UT value representation. DEFAULT: NO
ATT_0040E010_USE_UT_VR	In the 2014b edition of the DICOM Standard, the value representation of attribute (0040,E010) Retrieve URI was changed from UT to the newly introduced UR. For backward compatibility, this parameter specifies that, when reading messages or file objects, the attribute is expected to have the old UT value representation. DEFAULT: NO
ATT_0074100A_USE_ST_VR	In the 2014b edition of the DICOM Standard, the value representation of attribute (0074,100A) Contact URI was changed from ST to the newly introduced UR. For backward compatibility, this parameter specifies that, when reading messages or file objects, the attribute is expected to have the old ST value representation. DEFAULT: NO
CALCULATE_DEFINED_LENGTH_FOR_CB	This parameter is applied when a registered callback function expects the data length to be provided to it and the data length is undefined. If the parameter is set to No, the undefined length will be passed as is to the callback function. If the parameter value is Yes, the toolkit will calculate the actual value of the data length before passing it tot the callback function. DEFAULT: NO
CALLBACK_MIN_DATA_SIZE	When using the **MCapplication.registerProvider()** call to store large data such as pixel data, this option specifies the minimum size of value for which the callback function should be used. This option was specifically added so pixel data contained in icons are not managed with a callback function. DEFAULT: 1
COMPRESSION_ALLOW_FRAGS	Configuration Parameter for `MC_Standard_Compressor`. The Pegasus libraries allow compressed image data to be returned as it continues to compress more image data. This may result in an image frame having one or more fragments. This is perfectly legal, however some viewers may not be able to display the image if they do not support multiple fragments per frame. DEFAULT: YES

Name	Description
COMPRESSION_CHROM_FACTOR	Configuration Parameter for `MC_Standard_Compressor`. Values 0 through 255. The chrominance compression factor is used to adjust the default chrominance quantization table values. When ChromFactor is 32, the default chrominance quantization table values are used as is. A value of 255 corresponds to high compression, low quality. DEFAULT: 32
COMPRESSION_J2K_LOSSY_QUALITY	Configuration Parameter for `MCattributeSet.setCompression()`. When JPEG_2000 with COMPRESSION_WHEN_J2K_USE_LOSSY = Yes, and COMPRESSION_J2K_LOSSY_USE_QUALITY = Yes, a quality can be specified. Valid values are 1 to 10, 1 being highest quality image. DEFAULT: 1
COMPRESSION_J2K_LOSSY_RATIO	Configuration Parameter for `MCattributeSet.setCompression()`. When JPEG_2000 with COMPRESSION_WHEN_J2K_USE_LOSSY = Yes, and COMPRESSION_J2K_LOSSY_USE_QUALITY = No, a ratio can be specified. The compressor attempts to reduce the image size to 1/ COMPRESSION_J2K_LOSSY_RATIO. DEFAULT: 10
COMPRESSION_J2K_LOSSY_USE_QUALITY	Configuration Parameter for `MCattributeSet.setCompression()`. When JPEG_2000 with COMPRESSION_WHEN_J2K_USE_LOSSY = Yes, this indicates which metric should be used for lossy compression, ratio or quality. DEFAULT: YES
COMPRESSION_LUM_FACTOR	Configuration Parameter for `MCattributeSet.setCompression()`. Values 0 through 255. 0 is the highest quality, giving a quantization table of all 1's. 32 corresponds to the standard quantization tables. For values between 0 and 128, the standard tables are scaled linearly. For values between 128 and 255, the standard tables are scaled non-linearly and the compression increases (and the quality decreases) by a very large amount. DEFAULT: 32
COMPRESSION_RGB_TRANSFORM_FORMAT	This parameter allows the user to select the output format when doing Lossy JPEG compression of RGB images. The value can be set to YBR_FULL or YBR_FULL_422 to specifiy what photometric interpretion Merge DICOM Toolkit should compress into when compressing RGB images. DEFAULT: YBR_FULL_422

Name	Description
COMPRESSION_USE_HEADER_QUERY	If set to YES, it instructs the toolkit to give precedence to the image parameters (rows, columns, etc.) from the JPEG header, in case disagreement is suspected between the DICOM header the JPEG header. If set to NO, the DICOM header will be used. DEFAULT: NO
COMPRESSION_WHEN_J2K_USE_LOSSY	Configuration Parameter for `MCattributeSet.setCompression()`. When JPEG_2000 is used as a transfer syntax, this could mean either lossy or lossless compression. This parameter specifies the intended syntax. DEFAULT: No
CREATE_OFFSET_TABLE	This parameter specifies if an offset table is created when `MCattributeSet.duplicate()` is used to compress a DICOM message or file. It also specifies if an offset table is created when the `MCattributeSet.addEncapsulatedFrame()` method is used. DEFAULT: Yes
DECODER_TAG_FILTER	Specifies the list of tags to be ignored when reading DICOM files or messages. The values are separated by commas and can be specified in different formats: Single tag, e.g.: 00080020 Tag range, e.g.: 00080020-000800FF Single group, e.g.: G0020 Group range, e.g: G0020-G0022 All private as: PRIVATE All ranges are inclusive, meaning that G0020-G0022 will filter groups 20 and 22. DEFAULT: (empty)
DECODER_PRIVATE_TAG_WHITELIST	Specifies the list of private tags to be allowed during reading DICOM files or messages. The values are separated by commas and can bespecified in different formats: Single tag, e.g.: 00090001 Tag range, e.g.: 00090000-000900FF Single group, e.g.: G0021 Group range, e.g.: G0021-G0025 All ranges are inclusive, meaning that G0021-G0025 will allow all private groups in the range including 21 and 25. DEFAULT: (empty)
DEFLATE_ALLOW_FLUSH	Allows deflate to flush data occasionally to limit buffering. DEFAULT: Yes

Name	Description
DEFLATE_COMPRESSION_LEVEL	Allows the compression level of deflate to be specified when using deflated explicit VR little endian transfer syntax. 0 is no compression, 1 is fastest, and 9 compresses best. DEFAULT: -1
DESIRED_LAST_PDU_SIZE	This parameter allows the user to configure the length of the last PDU sent. This allows for interoperability with other DICOM implementations that may be intolerant with either a zero or two byte final PDU length. The default value used is 8. **Note:** Starting with release 3.5.1, this configuration option has a limited effect.
DICTIONARY_ACCESS	This parameter specifies whether or not the DICOM dictionary is to be loaded into memory or accessed from the dictionary file. FILE means access information directly from the dictionary file. MEM means load the dictionary into memory and access it there. **Note:** Starting with the 3.5.1 Merge DICOM Toolkit release, dictionary access is always memory based and can no longer be file based. This option is now ignored. DEFAULT: MEM
DICTIONARY_FILE	This parameter specifies the name (path) of the DICOM dictionary. An absolute or relative path may be specified. **Note:** This parameter is ignored if the dictionary has been pre-compiled. The path to the file can also be specified using environment variables (including the pseudo environment variable MC3INIDIR which does not need to be set as the toolkit will resolve it internally to the directory where the merge.ini file resides). Unicode paths can now be specified through the environment variables. Both Windows style (%) and Unix style ($) notations for the environment variables are accepted. DEFAULT: ../mc3msg/mrgcom3.dct
DUPLICATE_ENCAPSULATED_ICON	When duplicating to an encapsulated transfer syntax, this configuration value specifies whether an ICON IMAGE SEQUENCE should also be encapsulated. DEFAULT: NO
ELIMINATE_ITEM_REFERENCE *	This parameter specifies the behavior of the message/item/file disposing method `MCattributeSet.dispose()`. If this parameter is set to YES, the above functions will search for references in every currently open object to delete when they encounter an item to free within an object. DEFAULT: NO.

Name	Description
EMPTY_PRIVATE_CREATOR_CODES	If set to NO, private creator codes contained in messages are not emptied when the `MC_Empty_Message()` or `MC_Empty_File()` function calls are made. DEFAULT: YES **Note:** These underlying C/C++ toolkit functions are not called in the Python toolkit.
EXPLICIT_VR_TO_UN_FOR_LENGTH_GT_64K	If set to YES, the toolkit will allow encoding in explicit VR of data elements whose VR is none of OB, OW, OL, OV, OD, OF, SQ or UT and whose value length exceeds 65534 bytes by effectively changing the VR to UN (as per CP-1066). If set to NO, the attempt to encode such date elements will result in an `MC_INVALID_LENGTH_FOR_VR` error. DEFAULT: NO
EXPORT_EMPTY_PRIVATE_CREATOR_CODES	If set to NO it prevents the toolkit from exporting private creator data elements which don't have any private attributes in the private block. If set to YES, exporting private creator data elements with empty private blocks is allowed. DEFAULT: YES
EXPORT_GROUP_LENGTHS_TO_NETWORK *	When set to NO, do not export group length attributes when using the `sendRequestMessage()`, `sendRequestStorage()`,`sendResponseMessage()` and `sendResponseStorage()` methods of the `MCassociation` class. DEFAULT: YES
EXPORT_PRIVATE_ATTRIBUTES_TO_NETWORK	When set to NO, disable the exporting of private attributes in messages written to the network with the `sendRequestMessage()`, `sendRequestStorage()`, `sendResponseMessage()` and `sendResponseStorage()` methods of the `MCassociation` class. DEFAULT: YES
EXPORT_UN_VR_TO_NETWORK	When set to NO, disable the exporting of attributes with a VR of UN in messages written to the network with the `sendRequestMessage()`, `sendRequestStorage()`, `sendResponseMessage()` and `sendResponseStorage()` methods of the `MCassociation` class. DEFAULT: YES
EXPORT_UNDEFINED_LENGTH_SQ *	If YES, messages transferred over the network or written to disk have their sequence attributes encoded as undefined length. This increases performance of the library. DEFAULT: NO

Name	Description
FLATE_GROW_OUTPUT_BUF_SIZE *	The size that the output buffer of deflate or inflate should grow to when its size is insufficient. An Info message is logged each time the buffer grows. DEFAULT: 1024
FORCE_OPEN_EMPTY_ITEM *	When set to YES, the `MCitem` constructor will act similar to the `MCmessage` constructor. The up-front performance cost of the `MCitem` constructor will be reduced, but the amount of validation done when adding tags to the item is also reduced. Setting this value to YES will also improve the performance of the DICOMDIR directory functions. This configuration value does not have any effect on embedded platforms. DEFAULT: NO
IGNORE_JPEG_BAD_SUFFIX	Configuration Parameter for `MCattributeSet.setCompression()` to deal with lossless JPEG images whose suffix have been invalidly written according to the JPEG specification. These images have a 16-zero-bit suffix following a -32768 prefix where the JPEG spec says the suffix is omitted following a -32768 prefix. The following are the valid settings: -1 = Default, fail on these images 0 = Ignore when user detects such images 1 = Let the toolkit detect and ignore automatically
LARGE_DATA_SIZE	Defines "Large Data" to the toolkit. "Large Data" is defined as an attribute value which has a length of LARGE_DATA_SIZE or more. DEFAULT: 200.
LARGE_DATA_STORE	This parameter specifies where "Large Data" values should be stored. FILE means store the values in temporary files. MEM means store the values in memory. **Note:** Embedded systems should ignore this parameter and always use MEM. DEFAULT: MEM
LIST_SQ_DEPTH_LIMIT	Limit the depth of sequences listing. This parameter should be set to the maximum number of levels any sequence should be listed. DEFAULT: is 0 - means do not limit the listing of sequences
LIST_UN_ATTRIBUTES	If No, attributes with Unknown VR will not be listed by `MCattributeSet.list()` and T2 logging option. DEFAULT: Yes

Name	Description
LIST_VALUE_LIMIT	Limit the size of listed values by `MCattributeSet.list()` or T2 logging option. This parameter should be set to the maximum number of lines to be printed for any attribute in the list. DEFAULT: 0 - means show the whole value.
MSG_FILE_ITEM_OBJ_TRACE	This parameter allows the tracking of the creation, referencing and freeing of message, file and item objects. This option can be used if the user suspects a memory leak in their application from not freeing one of these object types. The logging is done at the T1 trace level which must be enabled in the merge.ini file. DEFAULT: NO
MSG_INFO_FILE	This parameter specifies the name (path) of the DICOM message information file. An absolute or relative path may be specified. The path to the file can also be specified using environment variables (including the pseudo environment variable MC3INIDIR which does not need to be set as the toolkit will resolve it internally to the directory where the merge.ini file resides). Unicode paths can now be specified through the environment variables. Both Windows style (%) and Unix style ($) notations for the environment variables are accepted. **Note:** This parameter is ignored if the message database has been pre-compiled. DEFAULT: ../mc3msg/mrgcom3.msg
NULL_TYPE3_VALIDATION	This parameter specifies how the toolkit will validate a single NULL value in a type 3 attribute with VM > 1. Valid values are ERR, WARN and INFO. DEFAULT: ERR
OBOW_BUFFER_SIZE	This parameter specifies the number of bytes of "Large Data" that should be buffered before they are written to disk. This value is only used when the parameter LARGE_DATA_STORE is set to FILE. DEFAULT: 4096
PEGASUS_DISP_REG_NAME	When using your own Pegasus license to remove the 3 frames/second limitation, this should have the company name that was used to generate your Pegasus license.
PEGASUS_DISP_REGISTRATION	When using your own Pegasus license to remove the 3 frames/second limitation, this should have the registration code that goes with the Pegasus dispatcher.

Name	Description
PEGASUS_NUMBER_OF_THREADS	Certain Pegasus opcodes can operate in a multithreaded manner. Use this setting to specify the number of threads to be used by the opcode. DEFAULT: 1
PEGASUS_OP_*_NAME	When using your own Pegasus license to remove the 3 frames/second limitation, this should have the company name that was used to generate your Pegasus license.
PEGASUS_OP_*_REGISTRATION	When using your own Pegasus license to remove the 3 frames/second limitation, this should have the registration code that goes with its respective PEGASUS_OP_*_NAME.
PEGASUS_OPCODE_PATH	This parameter specifies the directory where Pegasus opcode DLLs are to be loaded from. The opcode DLL refers to files like picn6220 and not the dispatcher DLL picn20. If the option is empty, the SSM/DLL is loaded from the same directory as the dispatcher DLL. If these files are not found, opcode SSM/DLL is loaded using the directory order Windows uses when loading DLLs. The SSM/DLL is loaded from the current directory if '.' is specified. DEFAULT: (empty)
REJECT_INVALID_VR	This parameter specifies whether or not to reject invalid VR values in DICOM messages. If set to Yes, the parsing is aborted and the data set is rejected with a status of MC_INVALID_VR. This is useful in some scenarios when invalid attribute VR and length can result in runaway read/copy operations which may lead to crashes. DEFAULT: No
RELEASE_SQ_ITEMS	If set to NO, existing item IDs will not be freed when setting a null value or an empty value or a new value to a sequence attribute. Setting it to YES will allow sequence items that have no other references to be freed. DEFAULT: No
REMOVE_PADDING_CHARS	When set to Yes, Merge DICOM Toolkit will remove space padding characters from all text based attributes. This removal will occur when the attribute is encoded with `MCattributeSet.setValue()`, or when the attribute is read with one of the streaming or network read methods. DEFAULT: No
REMOVE_SINGLE_TRAILING_SPACE	If set to YES, the toolkit will strip a single trailing padding space character from an attribute value of string type. Otherwise, it will not. DEFAULT: YES

Name	Description
RETURN_COMMA_IN_DS_FL_FD_STRINGS	When set to Yes, Merge DICOM Toolkit will return a comma character as a decimal separator in a value when `MCattributeSet.getStringValue()` is called for an attribute with a VR of DS, FL, or FD. When set to No, a period will always be returned for the decimal separator. Note that DS values will always be properly encoded with a period in DICOM message objects. DEFAULT: No
TEMP_FILE_DIRECTORY	This parameter specifies the directory in which temporary files should be created. This parameter is used only if LARGE_DATA_STORE = FILE. An absolute or relative path may be specified. The path to the directory can also be specified using environment variables (including the pseudo environment variable MC3INIDIR which does not need to be set as the toolkit will resolve it internally to the directory where the merge.ini file resides). Unicode paths can now be specified through the environment variables. Both Windows style (%) and Unix style ($) notations for the environment variables are accepted. DEFAULT: ./
TOLERATE_INVALID_IN_DEFAULT_CHARSET	This parameter specifies if non-ASCII characters are to be tolerated in the default repertoire. When set to Yes, the validation of the attribute/message will not be enforced, but a warning message will still be logged. DEFAULT: Yes
UN_VR_CODE	VR Code to use for attributes with unknown VRs. This may be set to 'OB' if an implementation does not understand 'UN'. DEFAULT: UN VALID VALUES: UN, OB
UPDATE_GROUP_0028_ON_DUPLICATE	When set to Yes, the group 0028 attributes within a message will be updated when duplicating a message or file with `MCattributeSet.duplicate()` and the standard compressor or decompressor. The Photometric Interpretation will be updated as appropriate, and the Lossy Image Compression, Lossy Image Compression Ratio and Lossy Image Compression Method tags will be updated if Lossy Image Compression was applied to the image. DEFAULT: No

Name	Description
USE_FREE_DATA_CALLBACK	When set to Yes, all `MCIstorageProviders` registered with `MCapplication.registerProvider()` are called with the `__FREE_DATA__` callback type when the memory associated with the callback is to be freed, because the enclosing message, file, or item is being freed. DEFAULT: No
WORK_BUFFER_SIZE *	This parameter specifies the amount of data that is buffered in the toolkit before being stored internally or passed to a user's callback function. This option impacts the following methods: - the `sendRequestMessage()`, `sendRequestStorage()`, `sendResponseMessage()`, `sendResponseStorage()`, `read()`, `continueRead()`, `readToStream()`, `continueReadToStream()` of class `MCassociation` - the `writeMessageToStream()` and `readMessageFromStream()` of class `MCattributeSet` and - the `readP10File()`, `readP10FileBypassBulk()`, `readP10Stream()`, `writeP10File()` and `writeP10FileToStream()` of class `MCfile`. Setting this option to values larger than 28K will in most cases cause the toolkit to use the operating system's memory management scheme instead of the toolkit's internal mechanism. DEFAULT: 28K

* Performance tuning

Table D.11: [TRANSPORT_PARMS] section of system profile parameters

Name	Description
CAPTURE_FILE	This parameter specifies the base name to use for capture files. (Capture files are generated if the NETWORK_CAPTURE value is set to Yes.) If only one capture file is requested (see NUMBER_OF_CAP_FILES), the capture file will have the name specified. If more than one is requested, nnn will be appended to the base file name specified (e.g., merge001.cap) DEFAULT: merge.cap (in the current directory) **Note:** Use of this parameter is deprecated.

Name	Description
CAPTURE_FILE_SIZE	This parameter specifies the maximum size (in kilobytes) that capture files are allowed to grow (capture files are generated if the NETWORK_CAPTURE value is set to Yes). If more than one capture file is requested (see NUMBER_OF_CAP_FILES), each file generated will have this maximum size. If a value less than 1 is specified only one capture file of unlimited length will be generated. DEFAULT: 0 **Note:** Use of this parameter is deprecated.
IP_TYPE	This parameter specifies the preferred IP type for network communications. When set to IPV4, Merge DICOM Toolkit will attempt to utilize only IPV4 network connections. When set to IPV6, Merge DICOM Toolkit will attempt to use only IPV6 network connections. When set to AVAILABLE in an SCP, Merge DICOM Toolkit will prefer IPV6 if it is enabled in the operating system over IPV4. If IPV6 is used, the socket is put into dual stack mode, if supported by the operating system, to accept connections from both IPV4 and IPV6. When set to AVAILABLE in an SCU, Merge DICOM Toolkit will use the available type of IP networking. DEFAULT: AVAILABLE VALID VALUES: AVAILABLE, IPV4, IPV6
MAX_PENDING_CONNECTIONS	This parameter specifies the maximum number of open listen channels. Its value is used as the second argument of a TCP listen() call. DEFAULT: 5
NETWORK_CAPTURE	This parameter specifies whether or not network data should be captured in files suitable to be read by the MergeDPM utility. Use these parameters to customize the network capture: CAPTURE_FILE CAPTURE_FILE_SIZE NUMBER_OF_CAP_FILES REWRITE_CAPTURE_FILES DEFAULT: No **Note:** Use of this parameter is deprecated.
NUMBER_OF_CAP_FILES	This parameter specifies the number of capture files to generate (capture files are generated if the NETWORK_CAPTURE value is set to Yes). Each capture file generated will have maximum size specified by CAPTURE_FILE_SIZE. If CAPTURE_FILE_SIZE is less than 1 (unlimited size) this parameter's value is ignored. DEFAULT: 1 **Note:** Use of this parameter is deprecated.
REWRITE_CAPTURE_FILES	This parameter specifies whether or not the capture files should be rewritten when all files have reached the maximum size specified by CAPTURE_FILE_SIZE (capture files are generated if the NETWORK_CAPTURE value is set to Yes). If Yes is specified, the oldest file will be rewritten. If No is specified and all requested files have been written (see NUMBER_OF_CAP_FILES), no more data will be captured. DEFAULT: Yes **Note:** Use of this parameter is deprecated.

Name	Description
TCPIP_DISABLE_NAGLE	This parameter specifies if the Nagle Algorithm should be used when sending packets at the TCP/IP level. Most operating systems enable this by default. It allows small segments of data to delay sending a fixed amount of time to possibly be combined with other small segments and be sent as one larger packet. Disabling this may cause high network traffic. DEFAULT: No
TCPIP_LISTEN_PORT	This parameter specifies the TCP/IP port on which server applications are to listen for associate requests. DEFAULT: 104
TCPIP_RECEIVE_BUFFER_SIZE *	This parameter specifies the TCP/IP receive buffer size for each connection. Note that the maximum values for this constant and TCPIP_SEND_BUFFER_SIZE are operating system dependent. If the values of these options are set too high, a message will be logged to the toolkit's log files, although no errors will be returned through the toolkit's API. Larger values for these constants will greatly improve network performance on networks with minimal network activity. Note that for optimum performance, these values should be at least slightly larger than the PDU_MAXIMUM_LENGTH configuration value. Note also that setting these values to an even multiple of the TCP/IP MSS (Maximum Segment Size) of 1460 bytes can help increase performance. Note, also that some operating systems such as Linux have auto-tuning of TCP/IP buffer sizes implemented when an explicit TCP/IP Send and Receive buffer size are not set. These options can be set to zero to disable Merge DICOM Toolkit's setting of each buffer size. DEFAULT: 131400 MAXIMUM: Operating System dependent
TCPIP_SEND_BUFFER_SIZE *	This parameter specifies the TCP/IP send buffer size for each connection. Note that the maximum values for this constant and TCPIP_RECEIVE_BUFFER_SIZE are operating system dependent. If the values of these options are set too high, a message will be logged to the toolkit's log files, although no errors will be returned through the toolkit's API. Larger values for these constants will greatly improve network performance on networks with minimal network activity. Note that for optimum performance, these values should be at least slightly larger than the PDU_MAXIMUM_LENGTH configuration value. Note also that setting these values to an even multiple of the TCP/IP MSS (Maximum Segment Size) of 1460 bytes can help increase performance. Note, also that some operating systems such as Linux have auto-tuning of TCP/IP buffer sizes implemented when an explicit TCP/IP Send and Receive buffer size are not set. These options can be set to zero to disable Merge DICOM Toolkit's setting of each buffer size. DEFAULT: 131400 MAXIMUM: Operating System dependent

* Performance tuning

D.4. Service Profile

The Service Profile is generated by Merge OEM and contains DICOM standard services and commands and is a useful reference (along with the `message.txt` file mentioned previously) to find the Merge DICOM names for the standard DICOM services and items. It is used by the library to negotiate the proper SOP Class UIDs and to access the binary dictionary and message information files when creating instances of message objects and validating messages.

In most cases, it will not be necessary to modify the Service Profile. However, if you are using an extended toolkit to create your own private services, you will need to add specifications for these private services to the Service Profile. See the Merge DICOM Toolkit: Extended Toolkit Manual for further details.

The location of the Service Profile is provided by the `MERGECOM_3_SERVICES` parameter of the `[MergeCOM3]` section of the `MERGE.INI` file.

Remember, the Service Profile is GENERATED by the Merge DICOM Profile Database Utilities at Merge OEM. Unless you are absolutely confident about changes being made, DO NOT CHANGE THE CONTENTS OF THIS FILE.

The Service Profile contains the following sections.

Table D.12: Service profile parameters

Name	Description
[SERVICE_TABLE]	List of service names and numbers. This list registers every service available to an Application Entity. The parameters associated with [SERVICE_LIST] are NUMBER_OF_SERVICES_SUPPORTED (the number of service names that will be listed immediately following NUMBER_OF_SERVICES_SUPPORTED) and one entry for each supported service.
[<service_number>]	One section number for each of the above services registered in [SERVICE_TABLE]. Each section contains a Service Name, a DICOM SOP Class UID for the Service, a flag that tells whether it is a BASE or META Service (SOP) and a list of commands supported for that service.
[ITEM_TABLE]	One item name and number for each DICOM item that can be encoded in an attribute of Value representation SQ (Sequence of Items).