



User Guide

Version 5.1.0
September 13, 2018

Table of Contents

UPDATE LIST	6
1. LICENSE AGREEMENT	7
2. INTRODUCTION	8
2.1 GENERAL	8
2.2 FEATURES	8
2.3 CAPABILITIES	9
2.4 RESTRICTIONS	9
2.4.1 General	9
2.5 DEFINITIONS/ABBREVIATIONS	9
2.6 REFERENCES	10
2.7 RELATIONSHIP WITH THE DICOM STANDARD	11
3. GETTING STARTED	12
3.1 SYSTEM REQUIREMENTS	12
3.2 RELEASE PACKAGE	12
3.3 INSTALLATION	14
4. DVT DATA STRUCTURE	15
4.1 GENERAL	15
4.2 PROJECT FILES	16
4.3 SESSION FILES	16
4.3.1 Emulator Sessions	16
4.3.2 Script Sessions	16
4.3.3 Media Sessions	16
4.4 DVT SESSION PROPERTIES	16
4.4.1 SUT Test Session Properties	17
4.4.2 SUT ACSE Properties	17
4.4.3 DVT ACSE Properties	18
4.4.4 Socket Properties	18
4.4.5 Test Session Properties	19
4.4.6 Supported Transfer Syntaxes	21
4.4.7 Definitions	22
4.4.8 DICOMScripts	22
4.4.9 Results	22
4.4.10 DICOMScript Description Directory	22
4.5 DEFINITION FILE	23
4.5.1 Standard Definition Files	23
4.5.2 Private Definition Files	27
4.5.3 Special Definition Files	27
4.6 DATA FILES	27
4.7 DICOM CHARACTER SETS - SUPPORTED BY DVTK	28
4.8 DICOMSCRIPT	29
4.9 DICOMSUPERSCRIPT	29
4.10 VBSCRIPT	30
4.11 RESULTS FILE	30
4.11.1 Global Results Files	31
4.12 MEDIA STORAGE FILE	31
4.13 RAW DATASET FILE	32
4.14 PIXEL FILE	32
4.15 FILE INDEX FILE	34
5. GRAPHICAL USER INTERFACE - GUI	35
5.1 SESSION TREE	37
5.2 TAB CONTROL	39
5.2.1 Session Information Tab	39
5.2.2 Specify SOP Classes Tab	44
5.2.3 Activity Logging Tab	45
5.2.4 Validation Results Tab	45
5.2.5 Script Tab	45
5.2.6 Results Manager Tab	46
5.3 MAIN MENU BAR	48
5.3.1 File Menu	48
5.3.2 Edit Menu	50

5.3.3	View Menu	51
5.3.4	Emulator Status Menu	52
5.3.5	Window Menu	53
5.3.6	Help Menu	53
5.4	TOOLBAR	54
5.4.1	New	54
5.4.2	Open	54
5.4.3	Save	54
5.4.4	Copy	54
5.4.5	Edit Script with Notepad	54
5.4.6	Find	54
5.4.7	Find Next Warning	54
5.4.8	Find Next Error	54
5.4.9	Stop	54
5.4.10	Navigate Back	55
5.4.11	Navigate Forward	55
5.5	CONTEXT MENU OF THE SESSION TREE	56
5.5.1	Session node	56
5.5.2	Script Session node	56
5.5.3	Media Session node	56
5.5.4	Script node	58
5.5.5	Results node	58
5.5.6	Emulator node	58
5.6	EMULATORS	59
5.6.1	Storage SCP Emulator	59
5.6.2	Storage SCU Emulator	60
5.6.3	Print SCP Emulator	60
5.6.4	Emulator Transfer Syntaxes	61
6.	COMMAND LINE	62
7.	PROGRAMMING DVT	64
7.1	GENERAL	64
7.2	SEND, RECEIVE ACSE REQUESTS AND RESPONSES	64
7.2.1	Associate Request	65
7.2.2	Associate Accept	66
7.2.3	Associate Reject	68
7.2.4	Release Request	68
7.2.5	Release Response	69
7.2.6	Abort Request	69
7.3	SEND, RECEIVE DICOM MESSAGES	70
7.3.1	General	70
7.3.2	Send, Receive	71
7.3.3	Sequence Syntax	74
7.4	PATTERN GENERATION	75
7.4.1	OB/OF/OW Pattern Generation	75
7.4.2	ST/LT/UT Pattern Generation	76
7.5	USING LABELS - VALUE MAPPING	76
7.6	VR KEYWORDS	77
7.7	EXTENDED CHARACTER SETS	78
7.8	ABSTRACT STORAGE SERVICE	78
7.9	ABSTRACT PRINT SERVICE	79
7.10	PRIVATE ATTRIBUTES	79
7.11	IMAGE (OBJECT) RELATIONSHIP ANALYSIS	80
7.12	REPLAY FEATURE	80
8.	INTERPRETING THE VALIDATION RESULTS	81
8.1	STATUS KEYWORDS	81
8.1.1	PASSED	81
8.1.2	FAILED	81
8.1.3	ERROR	81
8.1.4	WARNING	81
8.1.5	INFO (INFORMATION)	81
8.1.6	DEBUG	81
8.1.7	RELATION	81

8.2	CONDITIONAL ATTRIBUTE VALIDATION OUTPUT.....	82
8.3	OTHER REMARKS	83
9.	ADVANCED PROGRAMMING	84
9.1	DATA-WAREHOUSE	84
9.1.1	Create, Set and Delete	84
9.1.2	Import, Export.....	86
9.1.3	Read, Write	87
9.2	REUSE OF OBJECTS AND ATTRIBUTES.....	88
9.2.1	Compare	88
9.2.2	Confirm.....	88
9.2.3	Copy.....	89
9.2.4	Delay	89
9.2.5	Display	89
9.2.6	Echo.....	89
9.2.7	Reset.....	90
9.2.8	System	90
9.2.9	Time	90
9.2.10	Validate.....	90
9.2.11	Verbose	91
9.3	SCRIPT EXECUTION CONTEXT	91
9.3.1	Add Group Length.....	91
9.3.2	Application Entity	92
9.3.3	Define Sequence Length	92
9.3.4	Populate	92
9.3.5	Strict Validation	93
9.3.6	Validation	93
10.	VISUAL BASIC SCRIPTS.....	94
10.1	ENTRY POINT.....	94
10.2	.NET INTERFACE DVTK AND DVTKDATA.....	94
10.3	SESSION VARIABLE	94
10.4	VISUAL STUDIO .NET 2005	95
11.	USING SECURE SOCKETS.....	96
11.1	OVERVIEW	96
11.2	SOME DEFINITIONS	96
11.3	OVERVIEW OF TLS/SSL.....	97
11.4	FILE FORMATS.....	97
11.4.1	Security Credentials File Format	97
11.4.2	Trusted Certificate File Format	98
11.4.3	Generated Certificate and Private Key File Formats.....	98
11.5	SUPPORTED CIPHER SUITES.....	98
11.6	SAMPLE CERTIFICATE AND CREDENTIALS FILES	98
12.	EXAMPLES.....	100
12.1	STORAGE SOP CLASS.....	101
12.2	QUERY SOP CLASS.....	102
12.3	RETRIEVE SOP CLASS	102
12.4	WORKLIST SOP CLASS	103
12.5	PERFORMED PROCEDURE STEP SOP CLASS.....	103
12.6	STORAGE COMMITMENT SOP CLASS.....	103
12.7	STRUCTURED REPORTING SOP CLASS.....	104
12.8	PRINT MANAGEMENT SOP CLASS.....	104
12.9	VERIFICATION SOP CLASS	104
13.	APPENDICES	105
13.1	DICOMSCRIPT LANGUAGE REFERENCE	105
13.1.1	ADD-GROUP-LENGTH ON / OFF	105
13.1.2	APPLICATION-ENTITY	105
13.1.3	COMPARE / COMPARE_NOT.....	106
13.1.4	CONFIRM.....	106
13.1.5	COPY.....	106
13.1.6	CREATE	106
13.1.7	DEFINE-SQ-LENGTH ON / OFF.....	107
13.1.8	DELAY	107
13.1.9	DELETE.....	107

13.1.10	DISPLAY	107
13.1.11	ECHO	107
13.1.12	EXPORT	108
13.1.13	IMPORT	108
13.1.14	POPULATE ON / OFF	108
13.1.15	READ	108
13.1.16	RECEIVE	108
13.1.17	RESET	108
13.1.18	SEND	109
13.1.19	SET	109
13.1.20	STRICT-VALIDATION ON / OFF	109
13.1.21	SYSTEM	109
13.1.22	TIME	110
13.1.23	VALIDATE	110
13.1.24	VALIDATION	110
13.1.25	VERBOSE ON / OFF	110
13.1.26	WRITE	110
13.2	DEFINITION FILE FORMAT REFERENCE	111
13.3	ADVT COMPATIBILITY NOTES	115
13.3.1	ADVT Emulator Sessions	115
13.3.2	ADVT Definitions	115
13.3.3	Backslash Handling	115
13.4	THIRD PARTY COPYRIGHT NOTICES	116
13.4.1	UC Davis DICOM Library	116
13.4.2	OpenSSL	116

Update list

Version	Date	Description
2.1	June 03, 2005	Updated document to conform to DVT V 2.1 software release.
2.2	Feb 06, 2009	Updated document to conform to DVT V 2.4 software release.
2.4	Apr 06, 2009	Updated document to conform to DVT V 2.4 software release.
2.5	May 16, 2017	Releasing version 5.0.0 based on .NET Framework 4.
2.6	September 13, 2018	New transfer syntaxes are supported

1. License Agreement

The User is referred to the web site www.dvtk.org for details of the DVTK license agreement.

2. Introduction

2.1 General

DICOM is increasingly being used as the standard communication mechanism when integrating various medical products in a hospital environment. The medical products involved include Modalities (CT, MR, X-Ray, etc.), Workstations, Archives, Printers and HIS/RIS devices.

The intention of DICOM is to define the communication capabilities of each product type to a degree that allows products, supplied by different vendors, to be connected together to form an open, integrated diagnostic / treatment capability.

A need, therefore, exists to ensure that a product conforms to the DICOM standard in a way appropriate to its function. The DICOM Validation Tool (DVT) is a software utility that will assist in testing DICOM conformance.

This document is the User Guide for the DICOM Validation Tool.

It is assumed that the reader has a basic understanding of the DICOM standard [1].

Throughout this document the product being validated by DVT is known as the “System under Test” or SUT.

2.2 Features

DVT is used to provide an independent measurement of the accuracy of a Systems DICOM Interface, according to both the DICOM Standard and the SUT’s Conformance Statement.

DVT can be used as either SCU or SCP with a direct connection to the SUT (via TCP/IP). DVT acts as an emulator for the DICOM Service classes being tested. DVT can also create and validate DICOM media files.

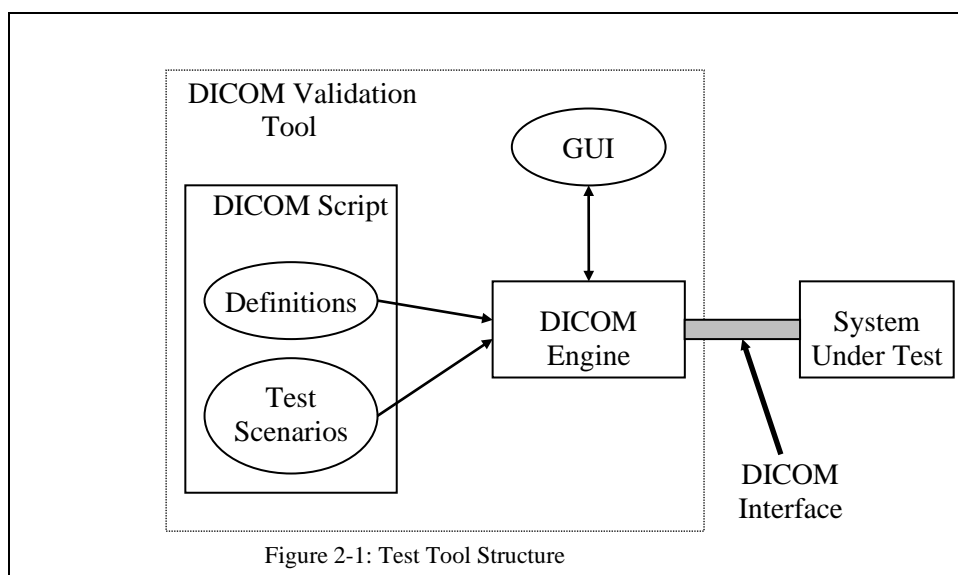


Figure 2-1: Test Tool Structure

The accuracy is measured in terms of syntax and semantic correctness. Complete inter-operability testing falls outside of the capabilities of such a validation tool.

2.3 Capabilities

- Data driven - tool core is independent of specific DICOM SOP Classes and Test Cases - allows future extension as the DICOM Standard evolves.
- Capable of playing the role of SCU and SCP.
- Generate / send and receive / validate DICOM encoded messages appropriate to the SOP Class being tested.
- The conditions specified in [1] are used for Module / Attribute validation where possible. During validation, when the condition cannot be evaluated logically, DVT displays the condition text to the User.
- Provides clear test results - including a summary of errors / warnings.
- High flexibility - access is provided to every single DICOM attribute and parameter.
- Received image data can be stored in Media Storage Format (See [1] - part 10) or in “raw” Dataset Format.
- Provides an Image (Object) Relationship Analysis for received objects.
- Allows tests to be repeated exactly as before - regression tests to check backward compatibility issues.
- Multi-platform support (Windows 2K / XP) as Command-line and GUI applications.
- Media File validation.
- DICOMDIR generation from a set of media files.
- Overview/Generation of test results report.
- Specific character set validation e.g. Japanese, Korean.
- Emulator for Verification, Storage and Print SOP Classes.
- Support for TLS and SSL secure sockets.
- .NET Assembly interface provides ability to write more complex test scripts in Visual Basic Script (VBScript).

2.4 Restrictions

2.4.1 General

DVT is incapable of testing all inter-operability issues. By concentrating on an interface validation, DVT has no way of checking what a SUT does with the transferred data. Some human interpretation will always be necessary.

DVT DICOM(Super)Scripts cannot handle receiving N-EVENT-REPORT messages at unknown times. All messages must be scripted in the exact order they are to be sent or received. If support for N-EVENT-REPORT messages is a requirement then the test script should be written in VBS with the appropriate actions being done on receipt of an N-EVENT-REPORT.

In general, older versions of DVT will not be able to read files created by newer versions of DVT. This includes Session Files, Definition Files and DICOM(Super)Scripts. As new versions of DVT are released, new features will be added to these files that the older versions of DVT will not support. The error messages generated by DVT will not always clearly identify why the file cannot be used.

The retired SOP class definition files provided by DVT are in “as-is” condition. They are not supported. Retired SOP class definition files are identified with RETIRED in the filename.

2.5 Definitions/Abbreviations

The following Definitions and Abbreviations are used throughout this project:

ACR	American College of Radiology.
ACSE	Association Control Service Element.

AE	Application Entity.
Command-line	Command line version of DVT.
DICOM	Digital Imaging and Communication in Medicine.
DIMSE	DICOM Message Service Element.
DVT	DICOM Validation Tool.
ELE	Explicit Little Endian.
EBE	Explicit Big Endian.
FSC	File-set Creator (Media).
FSR	File-set Reader (Media).
GUI	Graphical User Interface of DVT.
HIS	Hospital Information System.
HTML	Hyper-text Mark-up Language.
IOD	Information Object Definition.
ILE	Implicit Little Endian.
MS	Microsoft.
NEMA	National Electrical Manufacturers Association.
PDU	Protocol Data Unit.
RIS	Radiology Information System.
SCP	Service Class Provider.
SCU	Service Class User.
SOP	Service Object Pair.
SSL	Secure Sockets Layer.
SUT	System under Test - being tested by DVT for Conformance to the DICOM.
TLS	Transport Layer Security.
UID	Unique Identifier.
TCP/IP	Transmission Control Protocol/Internet Protocol.
TLS	Transport Layer Security.
TOC	Table of Content.
User	Person making use of DVT.
VBScript(VBS)	Visual Basic.NET Script Language.
VR	Value Representation.
XML	eXtensible Mark-up Language.
<INSTALLDIR>	Root directory where DVT is installed on the User's computer.

2.6 References

- [1] ACR/NEMA Standards Publications, No PS3, 1993-2004,
DICOM Standards:
Part 1 - Introduction,
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,
Part 10 - Media Storage and File Format for Media Interchange,
Part 11 - Media Storage Application Profiles,
Part 12 - Media Formats and Physical Media for Media Interchange,
Part 14 - Grayscale Standard Display Function,
Part 15 - Security Profiles,
Part 16 - Content Mapping Resource,
& various supporting Supplements.
- [2] DVT Conformance Statement,
Version 2.4

2.7 Relationship with the DICOM Standard

DVT has an “internal” knowledge of the ACSE and DIMSE message encoding. Software updates will be necessary if the way in which messages are encoded is changed in the DICOM Standard. It must be remembered that this is highly unlikely as it would affect all DICOM conforming products already in use.

Additions to the standard in the form of new Service Classes can be handled by DVT by generating a Definition File which corresponds to the new Service Class. DICOM (Super) Scripts will also be needed for any new Service Class. The Definition Files can also be modified if any change is made to any existing IOD content.

3. Getting started

3.1 System requirements

PC Requirements - Windows XP/7/8/10

- Pentium - 512 MB RAM.
- Ethernet - TCP/IP interface.
- CD ROM (for Media Validation).
- Internet connection (to install the .NET Framework 4 if needed).

3.2 Release package

A DVT release will include three installers:

- Tool software (EXE's (GUI, Command-line), assemblies, etc.) and supporting documentation.
- Set of standard SOP Class Definition Files (.DEF) – A separate installer.
- Set of example Test Scenarios (DICOM(Super)Scripts *.DS / *.DSS) – A separate installer.

The executables, standard Definition Files and Example will be released in the following directory structure:

- **<INSTALLDIR>**
 - **Bin**
Contains DVT executables and assemblies.
 - **Certificates**
Contains sample secure socket certificates and credentials.
 - **Docs**
Contains this document, the .NET Assembly Help Files and DVT Conformance Statement [2].
- **<%COMMONPROGRAMFILES%>\DVTk**
 - **Definition Files**
Contains all Definition Files.
- **My Documents\DVTk\Projects**
 - **Examples**
 - **Emulators**
Contains two emulators – either scu or scp.
 - **emulator_1**
 - **emulator_2**
 - **Media**
Contains Media Session File.
 - **Scripts**
 - **commit**
 - **scp**

- **scu**
- **mpps**
 - **scp**
 - **scu**
- **print**
 - **scp**
 - **scu**
- **query**
 - **scp**
 - **scu**
- **report**
 - **scp**
 - **scu**
- **retrieve**
 - **scp**
 - **scu**
- **storage**
 - **scp**
 - **scu**
- **verification**
 - **scp**
 - **scu**
- **worklist**
 - **scp**
 - **scu**

The script **scp** & **scu** directories contain example Test Scenarios for the appropriate service. Each **scp** or **scu** directory contains a **results** directory where the validation results files will be stored.

It is advised when generating SUT specific DICOM(Super)Scripts to do so in sub-directories relevant to the SOP Class being tested, i.e., if generating SCU SUT Worklist DICOM(Super)Scripts, do so in above directory structure.

The directory structure can be extended for other SOP Classes not released with DVT. The User will also need to generate Definition Files for these SOP Classes.

3.3 Installation

The DVT installation may require an Internet connection to install the Microsoft “**dotnetfx.exe**” if the .NET Framework 4 is not already installed on your computer.

To install DVT under Windows 2K/XP do the following:

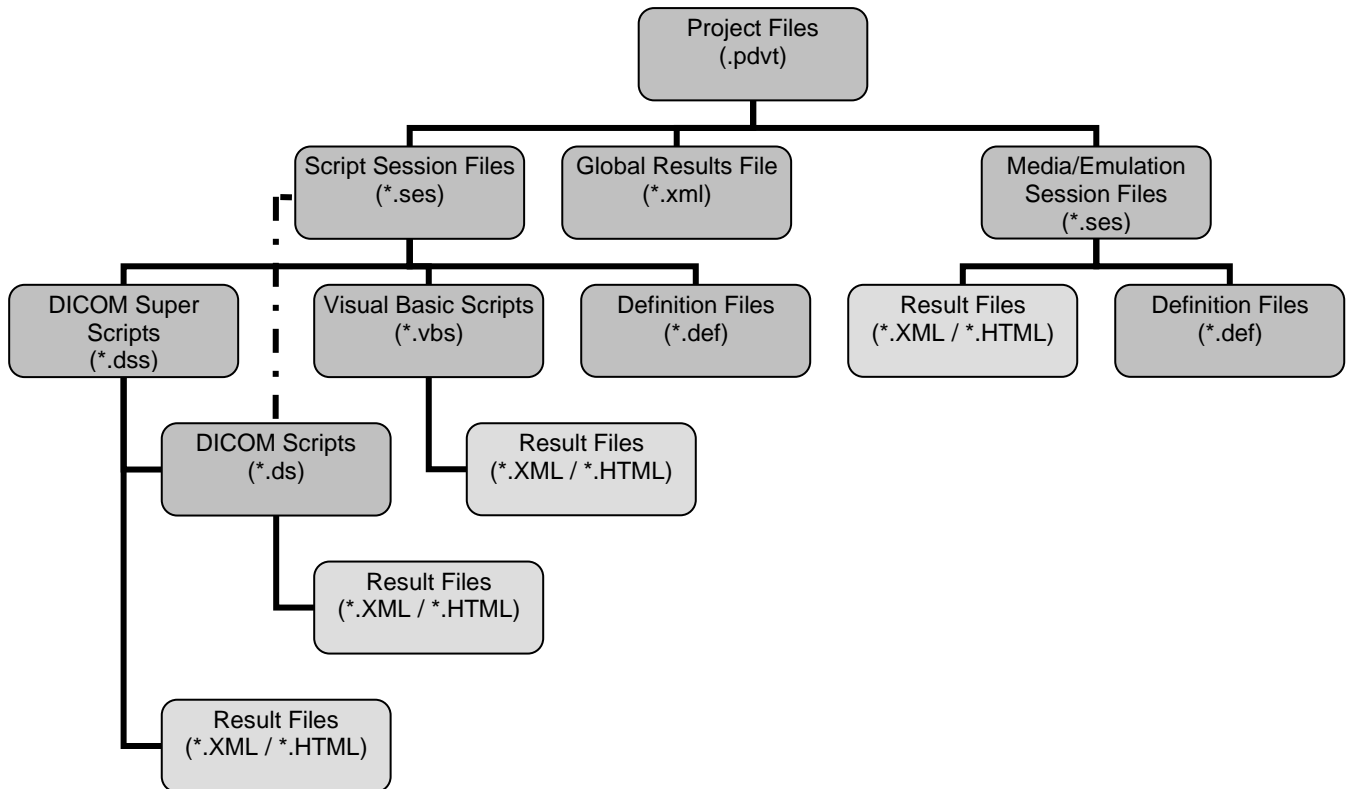
- Double click on the downloaded installer - this is a self-extracting .exe which will install DVT on your computer (XXX is current DVT 2.0 build number). Follow the given instructions.

An entry is automatically made for DVT (GUI), during installation in the “Program Files” menu option. Run DVT (GUI) by choosing this option. Open a Command Prompt window to run DVT (Command-line).

It is possible to associate .pdvt (Project Files) and .ses (Session Files) with the DVT GUI application (Windows “Open with...”). This will allow DVT to be started automatically when either a .pdvt file or .ses file is double-clicked.

Both GUI and Command-line versions of DVT are now ready for use.

4. DVT Data Structure



4.1 General

DVT makes use of the following files:

File Name	File Extensions
DVT Project files	*.PDVT
Session Files	*.SES
Definition Files	*.DEF
Data Files	*.DAT
DICOM Scripts	*.DS
DICOM Super Scripts	*.DSS
Visual Basic Scripts	*.VBS

The files are all ASCII text files that can be easily modified by the User. Each file type has a given format as described in the following sections.

DVT generates the following files:

File Name	File Extensions
Result Files	*RES.xml *RES.html
Pixel Data Files	*.PIX
“Raw” Dataset Files	*.RAW
Media Storage Files	*.DCM
Image Index Files	*.IDX

The *RES.xml are transformed into corresponding *RES.html files during display by the DVT GUI. The XSLT style sheet used for this transformation is provided in the <INSTALLDIR>\bin directory.

4.2 Project files

The concept of a DVT Project has been introduced in V2.0. A DVT project is a container for a number of DVT Sessions that are related to each other.

- A Project file is a collection of sessions.
- Within the DVT application only one project file can be opened at a time.
- Several “views” can be created (this makes it possible to run several sessions at the same time).
- Creates test reports at project level.

4.3 Session files

The concept of a DVT Session applies to the User’s activities when performing a DICOM Validation test of a given SUT. The Session is a very useful container for all configuration settings required to perform a set of tests on the SUT.

DVT supports three different Session types:

- Emulator (SCU or SCP).
- Script (script based).
- Media.

4.3.1 Emulator Sessions

An emulator session is used when DVT should act as an SCU or SCP emulator. DVT currently has emulator support as follows:

- Verification SCU
- Verification SCP
- Storage SCU
- Storage SCP (including Storage Commitment SCP)
- Print SCP

The DVT configuration parameters used during an emulator session are stored in an emulator Session File.

4.3.2 Script Sessions

A script session is used when DVT is used to execute DICOM(Super)Scripts and VBScripts. DVT currently has scripting support for:

- Network SCU and SCP message exchange – DVT can act as either SCU or SCP in the communication.
- Media file creation (FSC) – DICOM(Super)Scripts can be used to create media files on known content. These can be used to test media readers (FSR).

The DVT configuration parameters used during a script session are stored in a script Session File.

4.3.3 Media Sessions

Media sessions are used when media files (DICOMDIR, DCM, Part-10 files) are being validated. When validating a DICOMDIR file any referenced files found in the DICOMDIR will also be validated.

The DVT configuration parameters used during a media session are stored in a media Session File.

4.4 DVT Session properties

The Session filename can be composed of an alphanumeric part with a .SES extension.

The following parameters are supported:

- SUT Test Session Properties
- SUT ACSE Properties
- DVT ACSE Properties
- Socket Properties
- Test Session Properties
- Supported Transfer Syntaxes

- Definitions
- DICOMScripts
- Results
- DICOMScript Description Directory

4.4.1 SUT Test Session Properties

The SUT Test Session Properties are used to detail the Session specific properties and are:

- **SESSION-TYPE** *<string>(script, emulator, media)*
- **SESSION-FILE-VERSION** *<integer>*
- **SESSION-TITLE** *<string>*
- **SESSION-ID** *<integer> (1..9999)*
- **MANUFACTURER** *<string>*
- **MODEL-NAME** *<string>*
- **APPLICATION-ENTITY-NAME** *<string>*
- **APPLICATION-ENTITY-VERSION** *<string>*
- **SOFTWARE-VERSIONS** *<string>*
- **TESTED-BY** *<string>*
- **DATE** *<string> (YYYYMMDD)*

The **SESSION-TYPE** defines the type of session being one of script, emulator or media.

The **SESSION-FILE-VERSION** defines the version number of the session file.

The **SESSION-TITLE** is a free text string which can be used for a logical description of the session

The **SESSION-ID** is a number, ranging from 1 to 9999, used to identify the test session. The **SESSION-ID** is used as part of the Results File(s) name. This means that it is possible to execute a test case several times, each time changing the **SESSION-ID**, and have multiple test results available for the same test case.

The **MANUFACTURER** is the name of the company producing the Product being tested.

The **MODEL-NAME** is the Manufacturer's name for the Product being tested.

The **APPLICATION-ENTITY-NAME** is the name of the application (within the Product) being tested. The default value is "DICOM".

The **APPLICATION-ENTITY-VERSION** is the version of the application (within the Product) being tested. The default value is "3.0".

The **SOFTWARE-VERSIONS** is the Manufacturer's description of the software version(s) used by the Product being tested.

The **TESTED-BY** is the name of the person(s) performing the validation test.

The **DATE** is the date on which the validation test was performed. It must be expressed in YYYYMMDD format.

4.4.2 SUT ACSE Properties

The SUT ACSE Properties define the SUT parameters used when establishing an Association using the A-ASSOCIATE-RQ PDU and the A-ASSOCIATE-AC PDU and are:

- **SUT-ROLE** *<string>(requestor, acceptor)*
- **SUT-AE-TITLE** *<string>*
- **SUT-MAXIMUM-LENGTH-RECEIVED** *<integer>*

- **SUT-IMPLEMENTATION-CLASS-UID** <string>
- **SUT-IMPLEMENTATION-VERSION-NAME** <string>

The **SUT-ROLE** setting of *requestor* indicates that the SUT will take the role of Association Requestor and DVT the role of Association Acceptor in this session.

The **SUT-ROLE** setting of *acceptor* indicates that the SUT will take the role of Association Acceptor and DVT the role of Association Requestor in this session.

The **SUT-AE-TITLE** is the AE Title of the SUT in the test.

The **SUT-MAXIMUM-LENGTH-RECEIVED** is the maximum length of message fragment (P-DATA-TF PDU) that the SUT can receive from DVT. DICOM messages are split into P-DATA-TF PDU fragments - e.g., C-STORE-RQ of a modality image.

The **SUT-IMPLEMENTATION-CLASS-UID** is the DICOM UID assigned by the Manufacturer to the SUT implementation to identify it internally. DVT checks that the value sent by the SUT matches the value given here.

The **SUT-IMPLEMENTATION-VERSION-NAME** is the version name given by the Manufacturer to the SUT implementation to identify it internally. DVT checks that the value sent by the SUT matches the values given here. **NOTE:** *The IMPLEMENTATION-VERSION-NAME is an optional field - when the SUT does not send this value, enter "" (empty string) here.*

4.4.3 DVT ACSE Properties

The DVT ACSE Properties define the DVT parameters used when establishing an Association using the A-ASSOCIATE-RQ PDU and the A-ASSOCIATE-AC PDU and are:

- **DVT-AE-TITLE** <string>
- **DVT-MAXIMUM-LENGTH-RECEIVED** <integer>
- **DVT-IMPLEMENTATION-CLASS-UID** <string>
- **DVT-IMPLEMENTATION-VERSION-NAME** <string>

The **DVT-AE-TITLE** is the AE Title of DVT in the test.

The **DVT-MAXIMUM-LENGTH-RECEIVED** is the maximum length of message fragment (P-DATA-TF PDU) that DVT can receive from the SUT. DICOM messages are split into P-DATA-TF PDU fragments - e.g., C-STORE-RQ of a modality image.

The **DVT-IMPLEMENTATION-CLASS-UID** is the DICOM UID assigned to DVT.

The **DVT-IMPLEMENTATION-VERSION-NAME** is the version name assigned to DVT.

4.4.4 Socket Properties

The Socket Properties define the TCP/IP and the security properties and are:

- **SUT-HOSTNAME** <string>
- **SUT-PORT** <integer>
- **DVT-PORT** <integer>
- **DVT-SOCKET-TIMEOUT** <integer>
- **USE-SECURE-SOCKETS** <boolean> (true, false)
- **MAX-TLS-VERSION** <string> (TLSv1, TLSv1.1, TLSv1.2, TLSv1.3)

- **MIN-TLS-VERSION**<string> (TLSv1, TLSv1.1, TLSv1.2, TLSv1.3)
- **CHECK-REMOTE-CERTIFICATE** <boolean> (true, false)
- **CIPHER-LIST** <string>
- **CACHE-TLS-SESSIONS** <boolean> (true, false)
- **TLS-CACHE-TIMEOUT** <integer>
- **CREDENTIALS-FILENAME** <pathname>\<filename>
- **CERTIFICATE-FILENAME** <pathname>\<filename>

The **SUT-HOSTNAME** is the name that DVT should use when making a connection to the SUT. It is best to enter the Internet Address of the SUT (in dot notation).

The **SUT-PORT** is the port number that DVT should use when making a connection to the SUT.

The **DVT-PORT** is the port that the SUT should use when making a connection to DVT.

The **DVT-SOCKET-TIMEOUT** is the time-out in seconds used by DVT to wait for incoming data over a connected socket.

The **USE-SECURE-SOCKETS** is a boolean flag used to indicate whether the DICOM communication should be made over a secure socket (set **true**) or non-secure socket (set **false**).

The **MAX-TLS-VERSION** and **MIN-TLS-VERSION** indicates the max and min TLS version to be used. Only applicable when **USE-SECURE-SOCKETS** is true.

The **CHECK-REMOTE-CERTIFICATE** is a boolean flag used to indicate whether the X.509 certificate received from the SUT should be validated (set **true**) or not (set **false**) as part of the SSL Handshake. Only applicable when **USE-SECURE-SOCKETS** is true.

The **CIPHER-LIST** defines the cipher list used to establish the secure connection during the SSL Handshake. The value of the string is a concatenation of the possible settings for the SSL Authentication, Key Exchange, Data Integrity and Encryption parameters. Only applicable when **USE-SECURE-SOCKETS** is true.

The **CACHE-TLS-SESSIONS** is a boolean flag used to indicate whether the secure session established during the SSL Handshake should be cached for future use (set **true**) or not (set **false**). If the parameter is set true and the cache is still valid (see **TLS-CACHE-TIMEOUT**) then the SSL Handshake for any new connection is much shorter and the secure connection established more quickly. If the session cache is no longer valid then the full SSL Handshake is required to establish a new secure connection. Only applicable when **USE-SECURE-SOCKETS** is true.

The **TLS-CACHE-TIMEOUT** defines the time in seconds for which the secure session cache is valid. Only applicable when **USE-SECURE-SOCKETS** is true.

The **CREDENTIALS-FILENAME** defines the full pathname to a file containing the X.509 Private Key and Public Certificate that DVT should use when trying to establish a secure connection. Only applicable when **USE-SECURE-SOCKETS** is true.

The **CERTIFICATE-FILENAME** defines the full pathname to a file containing the X.509 Trusted Certificates of all the SUTs with which DVT will try to establish a secure connection. Only applicable when **USE-SECURE-SOCKETS** is true.

4.4.5 Test Session Properties

Test Session Properties are used to define switch settings for DVT. Some of these properties are flags indicating the level of detail in which to display the test results. The Test Properties are:

- **LOG-ERROR** <boolean> (true, false)
- **LOG-WARNING** <boolean> (true, false)
- **LOG-INFO** <boolean> (true, false)
- **LOG-RELATION** <boolean> (true, false)

- **LOG-DEBUG** <boolean> (**true**, **false**)
- **ODD Attribute Value Length**
- **LOG-DULP-STATE** <boolean> (**true**, **false**)
- **LOG-SCP-THREAD** <boolean> (**true**, **false**)
- **PDU-DUMP** <boolean> (**true**, **false**)
- **LABEL-DUMP** <boolean> (**true**, **false**)
- **STORAGE-MODE** <string> (**as-media**, **as-dataset**, **no-storage**)
- **CONTINUE-ON-ERROR** <boolean> (**true**, **false**)
- **STRICT-VALIDATION** <boolean> (**true**, **false**)
- **AUTO-TYPE-2-ATTRIBUTES** <boolean> (**true**, **false**)
- **AUTO-CREATE-DIRECTORY** <boolean> (**true**, **false**)
- **DEFINE-SQ-LENGTH** <boolean> (**true**, **false**)
- **ADD-GROUP-LENGTH** <boolean> (**true**, **false**)
- **DETAILED-VALIDATION-RESULTS** <boolean> (**true**, **false**)
- **SUMMARY-VALIDATION-RESULTS** <boolean> (**true**, **false**)
- **INCLUDE-TYPE-3-NOTPRESENT-INRESULTS** <boolean> (**true**, **false**)
- **Display Conditions in Summary Result**

The **LOG-ERROR** flag, when enabled (set **true**), causes any ERRORS detected by DVT to be logged.

The **LOG-WARNING** flag, when enabled (set **true**), causes any WARNINGS detected by DVT to be logged.

The **LOG-INFO** flag, when enabled (set **true**), causes any Information produced by DVT to be logged.

The **LOG-RELATION** flag, when enabled (set **true**), causes any DICOM Object Relationship information produced by DVT to be logged. The analysis shows the Patient/Study/Series/Object relationship between objects. The Patient ID (0010,0020), Study Instance UID (0020,000D), Series Instance UID (0020,000E) and SOP Instance UID (0008,0018) are used in the relationship analysis.

In addition, DVT displays (when available) the Patient Name (0010,0010), Referenced Image Sequence (0008,1140), Frame of Reference UID (0020,0052) for each image, together with the third value of the Image Type (0008,0008) (e.g., for CT this defines the image as a LOCALIZER (Scout) or an AXIAL image).

NOTE: DVT does not interpret whether the relationship is correct, i.e. that intended by the SUT - that must be done by the User according to the Test Scenario.

The **LOG-DEBUG** flag, when enabled (set **true**), causes any debug information to be logged.

The **LOG-DULP-STATE** flag, when enabled (set **true**), causes the event and state change information in the DULP Finite State Machine to be logged.

The **LOG-SCP-THREAD** flag, when enabled (set **true**), causes the SCP threads, created to handle the SCP emulations, to log data as described by the other LOG-xxx flags above.

The **PDU-DUMP** flag, when enabled, causes the sent / received messages to be displayed in BYTE format at the socket level, i.e., socket read & write. **NOTE:** Only, the first 1024 bytes of every P-DATA-TF PDU is displayed.

The **LABEL-DUMP** flag, when enabled, causes the final values of the LABEled attributes to be displayed to the user at the end of the DICOM(Super)Script.

The **STORAGE-MODE** flag can take one of the following values:

- **as-media** – Any received storage IOD will be stored in the DICOM Media Storage File format (i.e., including the File Prefix, DICOM Preamble and File Meta Information) with a file extension **.DCM**.
- **as-dataset** – Any received storage IOD will be stored in a “raw” format with only the Dataset being saved. The file extension is **.RAW**.
- **no-storage** – Do not store any received storage IOD.

The **CONTINUE-ON-ERROR** flag when enabled (set **true**), is used by DVT when executing DICOMSuperScripts, to continue the execution of the included DICOMScripts even if one of the DICOMScripts fails. When this flag is disabled DVT will stop DICOMSuperScript execution after the first occurrence of an error in a DICOMScript.

The **STRICT-VALIDATION** flag, when enabled (set **true**), causes DVT to perform a strict check between the received ACSE and DICOM messages and those programmed (expected) in the DICOMScripts. If any parameter of the received message does not match that programmed, then DVT reports a FAILED validation and aborts further DICOMScript interpretation.

If the **STRICT-VALIDATION** flag is disabled (set **false**), then DVT will WARN the User when a mismatch occurs between received and programmed values. DVT will continue interpreting the DICOMScript. This feature is particularly useful when the User wishes to test a specific (range of) SOP Class(es). Any additional SOP Classes proposed by the Product will be automatically rejected by DVT, allowing the User to concentrate on the SOP Class(es) of interest to the Test Scenario.

The **AUTO-TYPE-2-ATTRIBUTES** flag, when enabled (set **true**), causes DVT to add any Type 2 Attributes (with a zero-length) to the DICOMScript defined Dataset before sending it to the Product. DVT consults to Definition File corresponding to the Dataset in order to check the Dataset “completeness” before sending it to the Product. This feature ensures that the output produced by DVT conforms to the Dataset definition without the User having to explicitly set any zero-length Attributes.

The **AUTO-CREATE-DIRECTORY** flag, when enabled (set **true**), causes DVT to create any missing directories when handling the WRITE command to ensure that the pathname defined by the WRITE command is available before the file is written. This can be disabled to allow the user to generate the directories manually thus ensuring the intended directory structure is used.

The **DEFINE-SQ-LENGTH** flag, when enabled (set **true**), is used to make DVT encode explicit length Sequences when sending messages. Explicit lengths are computed for both the Sequence and each Item present within the Sequence. By default DVT uses the undefined length encoding.

The **ADD-GROUP-LENGTH** flag, when enabled (set **true**), is used to have DVT add Group Length attributes to all Groups found in each message sent. By default DVT does not encode Group Length attributes (except for the Command Group Length).

The **DETAILED-VALIDATION-RESULTS** flag, when enabled (set **true**), is used to have DVT produce detailed validation results which include all the attributes of all the modules present in any dataset being validated. This property can be used together with the SUMMARY-VALIDATION-RESULTS to get both detail and summary results during a test.

The **SUMMARY-VALIDATION-RESULTS** flag, when enabled (set **true**), is used to have DVT produce summary validation results. The summary only shows the attributes (in their modules) which have validation errors. This property can be used together with the DETAILED-VALIDATION-RESULTS to get both detail and summary results during a test.

The **INCLUDE-TYPE-3-NOTPRESENT-INRESULTS** flag, when enabled (set **true**), is used to have DVT include all Type 3 attributes in the detailed results even if these attributes are not present in the dataset being validated. When this property is disabled (set **false**) the detailed results produced contain only the attributes present in the dataset being validated.

4.4.6 Supported Transfer Syntaxes

The **SUPPORTED-TRANSFER-SYNTAX** <uid> parameter is used to define a Transfer Syntax that DVT will accept when working in an SCP emulation mode.

This parameter is repeated for each Transfer Syntax supported by DVT.

4.4.7 Definitions

The Session File can contain a **DEFINITION-ROOT** *<pathname>* parameter which is used to prefix the definition file names in order to define the full definition pathname.

The Session File can also contain an optional list of Definition Files. DVT loads this list of Definition Files when the Session File is opened. Each entry in the Session File is as follows:

DEFINITION *<pathname>\<filename>*

where *<pathname>* is the optional pathname for the given Definition File and *<filename>* is the name of the given Definition File. If the *<pathname>* is not present then the DEFINITION-ROOT will be used.

NOTE: DVT will automatically correct the pathname separator (slash or backslash) for the platform on which it is running. The User can use either type of separator.

The DEFINITION *<pathname>\<filename>* is repeated in the Session File for each Definition that must be loaded.

4.4.8 DICOMScripts

The Session File can contain a **DICOMSCRIPT-ROOT** *<pathname>* parameter which is used to prefix the DICOMScript names in order to define the full DICOMScript pathname.

The Session File can also contain an optional list of DICOMScripts. DVT will execute this list of DICOMScripts when the Session File is opened. Each entry in the Session File is as follows:

DICOMSCRIPT *<pathname>\<filename>*

where *<pathname>* is the optional pathname for the given DICOMScript and *<filename>* is the name of the given DICOMScript. If the *<pathname>* is not present then the DICOMSCRIPT-ROOT will be used.

NOTE: DVT will automatically correct the pathname separator (slash or backslash) for the platform on which it is running. The User can use either type of separator.

The DICOMSCRIPT *<pathname>\<filename>* is repeated in the Session File for each DICOMScript that must be executed.

4.4.9 Results

The Session File can contain a **RESULTS-ROOT** *<pathname>* parameter which is used to prefix the Result File names in order to define the full Result File pathname.

4.4.10 DICOMScript Description Directory

The **DESCRIPTION-DIRECTORY** property defines the directory in which HTML descriptions of the DICOMScripts can be found. The HTML description file has the same name as the DICOMScript but a HTML extension instead of .DS or .DSS.

The HTML description file can be used to provide a full description of the test scenario covered by the DICOMScript including graphics. The parameter has the following syntax:

DESCRIPTION-DIRECTORY *<pathname>*

A subset of the above parameters is found in the emulator, scripting and media Test Session files. The mapping between GUI property settings and Session File parameters is defined in the section on the GUI below.

4.5 Definition File

A Definition File (.DEF) describes a single DICOM (Meta) SOP Class in terms of the combination of DIMSE Commands and IOD's that make up the (Meta) SOP Class. The combination of DIMSE Commands and IOD's is taken from the DICOM Standard - parts 3 and 4.

The Definition Files provide DVT with the DICOM specific knowledge to enable a validation.

The Definition Files in this version of DVT support the MACRO syntax used in the DICOM Standard and describes the conditions under which type 1C and 2C attributes should be present.

The format of the Definition File is described in the section on the Definition File Format below. Definition files for the most frequently used DICOM (Meta) SOP Classes are provided as part of the DVT release package - see Tables 6.1 to 6.10.

4.5.1 Standard Definition Files

The following tables provide an overview of the most common DICOM (Meta) SOP Classes and the associated Definition Files (provided as part of the DVT release).

Table 6.1 - Storage SOP Class - Definition File Names

Storage SOP Class	SOP Class UID	Definition File Name (.def)
Computed Radiography Image	1.2.840.10008.5.1.4.1.1.1	ComputedRadiographyImageStorage
Digital X-Ray Image – for Presentation	1.2.840.10008.5.1.4.1.1.1.1	DigitalX-RayImageStorage-ForPresentation
Digital X-Ray Image – for Processing	1.2.840.10008.5.1.4.1.1.1.1.1	DigitalX-RayImageStorage-ForProcessing
Digital Mammography X-Ray Image – for Presentation	1.2.840.10008.5.1.4.1.1.1.2	DigitalMammographyX-RayImageStorage-Presentation
Digital Mammography X-Ray Image – for Processing	1.2.840.10008.5.1.4.1.1.1.2.1	DigitalMammographyX-RayImageStorage-Proprocessing
Digital Intra-Oral X-Ray Image – for Presentation	1.2.840.10008.5.1.4.1.1.1.3	DigitalIntra-oralX-RayImageStorage-Presentation
Digital Intra-Oral X-Ray Image – for Processing	1.2.840.10008.5.1.4.1.1.1.3.1	DigitalIntra-oralX-RayImageStorage-Proprocessing
CT Image	1.2.840.10008.5.1.4.1.1.2	CTImageStorage
Enhanced CT Image	1.2.840.10008.5.1.4.1.1.2.1	EnhancedCTImageStorage.def
Ultrasound Multi-frame Image	1.2.840.10008.5.1.4.1.1.3.1	UltrasoundMulti-frameImageStorage
MR Image	1.2.840.10008.5.1.4.1.1.4	MRImageStorage
Enhanced MR Image	1.2.840.10008.5.1.4.1.1.4.1	EnhancedMRImageStorage
MR Spectroscopy	1.2.840.10008.5.1.4.1.1.4.2	MRspectroscopyStorage
Ultrasound Image	1.2.840.10008.5.1.4.1.1.6.1	UltrasoundImageStorage
Secondary Capture Image	1.2.840.10008.5.1.4.1.1.7	SecondaryCaptureImageStorage
Multi-frame Single Bit Secondary Capture Image	1.2.840.10008.5.1.4.1.1.7.1	Multi-frameSingleBitSecondaryCaptureImageStorage
Multi-frame Grayscale Byte Secondary Capture Image	1.2.840.10008.5.1.4.1.1.7.2	Multi-frameGrayscaleByteSecondaryCaptureImageStorage
Multi-frame Grayscale Word Secondary Capture Image	1.2.840.10008.5.1.4.1.1.7.3	Multi-frameGrayscaleWordSecondaryCaptureImageStorage
Multi-frame True Color Secondary Capture Image	1.2.840.10008.5.1.4.1.1.7.4	Multi-frameTrueColorSecondaryCaptureImageStorage
Standalone Overlay	1.2.840.10008.5.1.4.1.1.8	StandaloneOverlayStorage
Standalone Curve	1.2.840.10008.5.1.4.1.1.9	StandaloneCurveStorage
12-lead ECG Waveform	1.2.840.10008.5.1.4.1.1.9.1.1	12-LeadECGWaveformStorage
General ECG Waveform	1.2.840.10008.5.1.4.1.1.9.1.2	GeneralECGWaveformStorage
Ambulatory ECG Waveform	1.2.840.10008.5.1.4.1.1.9.1.3	AmbulatoryECGWaveformStorage
Hemodynamic Waveform	1.2.840.10008.5.1.4.1.1.9.2.1	HemodynamicWaveformStorage

Table 6.1 - Storage SOP Class - Definition File Names

Storage SOP Class	SOP Class UID	Definition File Name (.def)
Cardiac Electrophysiology Waveform Storage	1.2.840.10008.5.1.4.1.1.9.3.1	CardiacElectrophysiologyWaveformStorage
Basic Voice Audio Waveform	1.2.840.10008.5.1.4.1.1.9.4.1	BasicVoiceAudioWaveformStorage
Standalone Modality LUT	1.2.840.10008.5.1.4.1.1.10	StandaloneModalityLUTStorage
Standalone VOI LUT	1.2.840.10008.5.1.4.1.1.11	StandaloneVOILUTStorage
Grayscale Softcopy Presentation State	1.2.840.10008.5.1.4.1.1.11.1	SoftcopyPresentationStateStorage
Color Softcopy Presentation State	1.2.840.10008.5.1.4.1.1.11.2	ColorSoftcopyPresentationStateStorage
Pseudo-Color Softcopy Presentation State	1.2.840.10008.5.1.4.1.1.11.3	Pseudo-ColorSoftcopyPresentationStateStorage
Blending Softcopy Presentation State	1.2.840.10008.5.1.4.1.1.11.4	BlendingSoftcopyPresentationStateStorage.def
X-Ray Angiographic Image	1.2.840.10008.5.1.4.1.1.12.1	X-RayAngiographicImageStorage
Enhanced X-Ray Angiographic Image	1.2.840.10008.5.1.4.1.1.12.1.1	EnhancedXAImageStorage
X-Ray Radiofluoroscopic Image	1.2.840.10008.5.1.4.1.1.12.2	X-RayRadiofluoroscopicImageStorage
Enhanced X-Ray Radiofluoroscopic Image	1.2.840.10008.5.1.4.1.1.12.2.1	EnhancedXRFIImageStorage
Nuclear Medicine Image	1.2.840.10008.5.1.4.1.1.12.2.1	NuclearMedicineImageStorage
Raw Data	1.2.840.10008.5.1.4.1.1.16	RawDataStorage
Spatial Registration	1.2.840.10008.5.1.4.1.1.16.1	SpatialRegistrationStorage
Spatial Fiducials	1.2.840.10008.5.1.4.1.1.16.2	SpatialFiducialsStorage
Deformable Spatial Registration	1.2.840.10008.5.1.4.1.1.16.3	DeformableSpatialRegistrationStorage.def
Segmentation	1.2.840.10008.5.1.4.1.1.16.4	SegmentationStorage.def
Real World Value Mapping	1.2.840.10008.5.1.4.1.1.16.7	RealWorldValueMappingStorage
VL Endoscopic Image	1.2.840.10008.5.1.4.1.1.17.1.1	VLEndoscopicImageStorage
Video Endoscopic Image	1.2.840.10008.5.1.4.1.1.17.1.1.1	VideoEndoscopicImageStorage
VL Microscopic Image	1.2.840.10008.5.1.4.1.1.17.1.2	VLMicroscopicImageStorage
Video Microscopic Image	1.2.840.10008.5.1.4.1.1.17.1.2.1	VideoMicroscopicImageStorage
VL Slide-Coordinates Microscopic Image	1.2.840.10008.5.1.4.1.1.17.1.3	VLSlide-CoordinatesMicroscopicImageStorage
VL Photographic Image	1.2.840.10008.5.1.4.1.1.17.1.4	VLPhotographicImageStorage
Video Photographic Image	1.2.840.10008.5.1.4.1.1.17.1.4.1	VideoPhotographicImageStorage
OphthalmicPhotography8BitImage	1.2.840.10008.5.1.4.1.1.17.1.5.1	OphthalmicPhotography8BitImageStorage
OphthalmicPhotography16Bit	1.2.840.10008.5.1.4.1.1.17.1.5.2	OphthalmicPhotography16BitImageStorage
Stereometric Relationship	1.2.840.10008.5.1.4.1.1.17.1.5.3	StereometricRelationshipStorage
Basic Text SR	1.2.840.10008.5.1.4.1.1.18.11	BasicTextSR
Enhanced SR	1.2.840.10008.5.1.4.1.1.18.22	EnhancedSR
Comprehensive SR	1.2.840.10008.5.1.4.1.1.18.33	ComprehensiveSR
Procedure Log Storage	1.2.840.10008.5.1.4.1.1.18.40	ProcedureLogStorage
Mammography CAD SR	1.2.840.10008.5.1.4.1.1.18.50	MammographyCADSR
Key Object Selection Document	1.2.840.10008.5.1.4.1.1.18.59	KeyObjectSelectionDocument
Chest CAD SR	1.2.840.10008.5.1.4.1.1.18.65	ChestCADSR
X-Ray Radiation Dose SR	1.2.840.10008.5.1.4.1.1.18.67	X-RayRadiationDoseSR
Positron Emission Tomography Image	1.2.840.10008.5.1.4.1.1.128	PositronEmissionTomographyImageStorage
Standalone PET Curve	1.2.840.10008.5.1.4.1.1.129	StandalonePETCurveStorage
RT Image	1.2.840.10008.5.1.4.1.1.481.1	RTImageStorage
RT Dose	1.2.840.10008.5.1.4.1.1.481.2	RTDoseStorage
RT Structure Set	1.2.840.10008.5.1.4.1.1.481.3	RTStructureSetStorage

Table 6.1 - Storage SOP Class - Definition File Names

Storage SOP Class	SOP Class UID	Definition File Name (.def)
RT Beams Treatment Record	1.2.840.10008.5.1.4.1.1.481.4	RTBeamsTreatmentRecordStorage
RT Plan	1.2.840.10008.5.1.4.1.1.481.5	RTPlanStorage
RT Brachy Treatment Record	1.2.840.10008.5.1.4.1.1.481.6	RTBrachyTreatmentRecordStorage
RT Treatment Summary Record	1.2.840.10008.5.1.4.1.1.481.7	RTTreatmentSummaryRecordStorage
RT Ion Plan Storage	1.2.840.10008.5.1.4.1.1.481.8	RTIonPlanStorage
RT Ion Beams Treatment Record Storage	1.2.840.10008.5.1.4.1.1.481.9	RTIonBeamsTreatmentRecordStorage
Media Storage Directory Storage	1.2.840.10008.1.3.10	MediaStorageDirectoryStorage

Table 6.2 - Query / Retrieve SOP Class – Definition File Names

Query / Retrieve SOP Class	SOP Class UID	Definition File Name (.def)
Patient Root QR – FIND	1.2.840.10008.5.1.4.1.2.1.1	PatientRootQueryRetrieve-FIND
Patient Root QR - MOVE	1.2.840.10008.5.1.4.1.2.1.2	PatientRootQueryRetrieve-MOVE
Patient Root QR - GET	1.2.840.10008.5.1.4.1.2.1.3	PatientRootQueryRetrieve-GET
Study Root QR – FIND	1.2.840.10008.5.1.4.1.2.2.1	StudyRootQueryRetrieve-FIND
Study Root QR – MOVE	1.2.840.10008.5.1.4.1.2.2.2	StudyRootQueryRetrieve-MOVE
Study Root QR – GET	1.2.840.10008.5.1.4.1.2.2.3	StudyRootQueryRetrieve-GET
Patient Study QR - FIND	1.2.840.10008.5.1.4.1.2.3.1	PatientStudyOnlyQueryRetrieve-FIND
Patient Study QR - MOVE	1.2.840.10008.5.1.4.1.2.3.2	PatientStudyOnlyQueryRetrieve-MOVE
Patient Study QR - GET	1.2.840.10008.5.1.4.1.2.3.3	PatientStudyOnlyQueryRetrieve-GET
Patient Root QR Relational - FIND	1.2.840.10008.5.1.4.1.2.1.1	PatientRootQRRelational-FIND
Patient Root QR Relational - MOVE	1.2.840.10008.5.1.4.1.2.1.2	PatientRootQRRelational-MOVE
Patient Root QR Relational - GET	1.2.840.10008.5.1.4.1.2.1.3	PatientRootQRRelational-GET
Study Root QR Relational - FIND	1.2.840.10008.5.1.4.1.2.2.1	StudyRootQRRelational-FIND
Study Root QR Relational - MOVE	1.2.840.10008.5.1.4.1.2.2.2	StudyRootQRRelational-MOVE
Study Root QR Relational - GET	1.2.840.10008.5.1.4.1.2.2.3	StudyRootQRRelational-GET
Patient Study QR Relational - FIND	1.2.840.10008.5.1.4.1.2.3.1	PatientStudyOnlyQRRelational-FIND
Patient Study QR Relational - MOVE	1.2.840.10008.5.1.4.1.2.3.2	PatientStudyOnlyQRRelational-MOVE
Patient Study QR Relational - GET	1.2.840.10008.5.1.4.1.2.3.3	PatientStudyOnlyQRRelational-GET

Table 6.3 - Print Management SOP Class – Definition File Names

Print Management SOP Class	SOP Class UID	Definition File Name (.def)
Basic Film Session	1.2.840.10008.5.1.1.1	BasicFilmSession
Basic Film Box	1.2.840.10008.5.1.1.2	BasicFilmBox
Basic Grayscale Image Box	1.2.840.10008.5.1.1.4	BasicGrayscaleImageBox
Basic Color Image Box	1.2.840.10008.5.1.1.4.1	BasicColorImageBox
Basic Grayscale Meta	1.2.840.10008.5.1.1.9	BasicGrayscalePrintManagementMeta
Print Job	1.2.840.10008.5.1.1.14	PrintJob
Basic Annotation Box	1.2.840.10008.5.1.1.15	BasicAnnotationBox
Printer	1.2.840.10008.5.1.1.16	Printer
Printer Configuration Retrieval	1.2.840.10008.5.1.1.16.376	PrinterConfigurationRetrieval
Basic Color Meta	1.2.840.10008.5.1.1.18	BasicColorPrintManagementMeta
Presentation LUT	1.2.840.10008.5.1.1.23	PresentationLUT
Image Overlay Box SOP Class (Retired)	1.2.840.10008.5.1.1.24	ImageOverlayBox
Basic Print Image Overlay Box (Retired)	1.2.840.10008.5.1.1.24.1	BasicPrintImageOverlayBox
Print Queue Management (Retired)	1.2.840.10008.5.1.1.26	PrintQueueManagement

Stored Print Storage (Retired)	1.2.840.10008.5.1.1.27	StoredPrintStorage
Hardcopy Grayscale Image Storage (Retired)	1.2.840.10008.5.1.1.29	HardcopyGrayscaleImageStorage
Hardcopy Color Image Storage (Retired)	1.2.840.10008.5.1.1.30	HardcopyColorImageStorage
Pull Print Request (Retired)	1.2.840.10008.5.1.1.31	PullPrintRequest
Pull Stored Print Management Meta (Retired)	1.2.840.10008.5.1.1.32	PullStoredPrintManagementMeta

Table 6.4 - Worklist SOP Class - Definition File Names

Worklist SOP Class	SOP Class UID	Definition File Name (.def)
Modality Worklist	1.2.840.10008.5.1.4.31	ModalityWorklist-FIND
General Purpose Worklist Management Meta	1.2.840.10008.5.1.4.32	GeneralPurposeWorklistManagementMeta
General Purpose Worklist	1.2.840.10008.5.1.4.32.1	GeneralPurposeWorklist-FIND
General Purpose Scheduled Procedure Step	1.2.840.10008.5.1.4.32.2	GeneralPurposeScheduledProcedureStep
General Purpose Performed Procedure Step	1.2.840.10008.5.1.4.32.3	GeneralPurposePerformedProcedureStep

Table 6.5 – Performed Procedure Step SOP Class - Definition File Names

PPS SOP Class	SOP Class UID	Definition File Name (.def)
Modality Performed Procedure Step	1.2.840.10008.3.1.2.3.3	ModalityPerformedProcedureStep
Modality PPS Retrieve	1.2.840.10008.3.1.2.3.4	ModalityPerformedProcedureStepRetrieve
Modality PPS Notification	1.2.840.10008.3.1.2.3.5	ModalityPerformedProcedureStepNotification

Table 6.6 - Detached Management SOP Class – Definition File Names

Detached SOP Class	SOP Class UID	Definition File Name (.def)
Detached Patient Management	1.2.840.10008.3.1.2.1.1	DetachedPatientManagement
Detached Patient Management Meta	1.2.840.10008.3.1.2.1.4	DetachedPatientManagementMeta
Detached Visit Management	1.2.840.10008.3.1.2.2.1	DetachedVisitManagement
Detached Study Management	1.2.840.10008.3.1.2.3.1	DetachedStudyManagement
Study Component Management	1.2.840.10008.3.1.2.3.2	StudyComponentManagement
Detached Results Management	1.2.840.10008.3.1.2.5.1	DetachedResultsManagement
Detached Results Management Meta	1.2.840.10008.3.1.2.5.4	DetachedResultsManagementMeta
Detached Study Management Meta	1.2.840.10008.3.1.2.5.5	DetachedStudyManagementMeta
Detached Interpretation Management	1.2.840.10008.3.1.2.6.1	DetachedInterpretationManagement

Table 6.7 – Storage Commitment SOP Class - Definition File Names

Commitment SOP Class	SOP Class UID	Definition File Name (.def)
Storage Commitment Push Model	1.2.840.10008.1.20.1	StorageCommitmentPush
Storage Commitment Pull Model	1.2.840.10008.1.20.2	StorageCommitmentPull

Table 6.8 – Study Content Notification SOP Class - Definition File Names

Notification SOP Class	SOP Class UID	Definition File Name (.def)
Basic Study Content Notification	1.2.840.10008.1.9	BasicStudyContentNotification

Table 6.9 - Verification SOP Class - Definition File Names

Verification SOP Class	SOP Class UID	Definition File Name (.def)
Verification	1.2.840.10008.1.1	Verification

Table 6.10 – Procedural Event Logging SOP Class - Definition File Names

Verification SOP Class	SOP Class UID	Definition File Name (.def)
Procedural Event Logging	1.2.840.10008.1.40	ProceduralEventLogging

4.5.2 Private Definition Files

The User can make Definition Files for any Private SOP Classes by taking one of the Standard Definition Files as a template. Private UID's, Modules and Attributes can be defined.

It is sometimes useful to take a Standard Definition File and customize it for a particular SUT. The Module / Attribute requirements can be made stricter, e.g., if the SUT is a modality which always provides a value for the Patient Name (0010,0010), then the User can modify the corresponding Definition File to make the Patient Name attribute Type 1 for this modality. DVT will then treat the Patient Name as a mandatory attribute during the SUT tests.

NOTE: When customizing definitions, it may be sensible to make the requirements for a particular SUT stricter (e.g., Type 2 can become Type 1, Conditionals can become Mandatory), but never make the requirements more relaxed (e.g., Type 1 becomes Type 2) as this will inevitably lead to a Standard violation.

4.5.3 Special Definition Files

There are several special definition files provided and used by DVT. These are described below.

DicomDirSpecial.def - used for Media creation and validation.

FileMeta.def – file Meta information.

4.6 Data Files

There are several data files provided and used by DVT. These are described below.

CharacterSets.dat – defines the character sets used for VR validation.

ImageDisplayFormat.dat - defines the number of image boxes belonging to a CUSTOM IMAGE-DISPLAY-FORMAT and the number of annotation boxes belonging to an ANNOTATION-DISPLAY-FORMAT-ID. Used by the Print SCP Emulator.

These Data Files are loaded from the <INSTALLDIR>\bin directory as DVT starts up.

4.7 DICOM Character Sets - supported by DVTk

Defined Terms for single-byte character sets without code extensions.			
Name	Defined Term	Code Element	Escape Sequence
<i>Default Repertoire</i>	None ISO_IR 6	G0	
<i>Latin alphabet No. 1</i>	ISO_IR 100	G0	
		G1	
<i>Latin alphabet No. 2</i>	ISO_IR 101	G0	
		G1	
<i>Latin alphabet No. 3</i>	ISO_IR 109	G0	
		G1	
<i>Latin alphabet No. 4</i>	ISO_IR 110	G0	
		G1	
<i>Cyrillic</i>	ISO_IR 144	G0	
		G1	
<i>Arabic</i>	ISO_IR 127	G0	
		G1	
<i>Greek</i>	ISO_IR 126	G0	
		G1	
<i>Hebrew</i>	ISO_IR 138	G0	
		G1	
<i>Latin alphabet No. 5</i>	ISO_IR 148	G0	
		G1	
<i>Japanese</i>	ISO_IR 13	G0	
		G1	
<i>Thai</i>	ISO_IR 166	G0	
		G1	

Defined Terms for single-byte character sets with code extensions.			
Name	Defined Term	Code Element	Escape Sequence
<i>Default Repertoire</i>	ISO 2022 IR 6	G0	ESC 28 42
<i>Latin alphabet No. 1</i>	ISO 2022 IR 100	G0	ESC 28 42
		G1	ESC 2D 41
<i>Latin alphabet No. 2</i>	ISO 2022 IR 101	G0	ESC 28 42
		G1	ESC 2D 42
<i>Latin alphabet No. 3</i>	ISO 2022 IR 109	G0	ESC 28 42
		G1	ESC 2D 43
<i>Latin alphabet No. 4</i>	ISO 2022 IR 110	G0	ESC 28 42
		G1	ESC 2D 44
<i>Cyrillic</i>	ISO 2022 IR 144	G0	ESC 28 42
		G1	ESC 2D 4C
<i>Arabic</i>	ISO 2022 IR 127	G0	ESC 28 42
		G1	ESC 2D 47
<i>Greek</i>	ISO 2022 IR 126	G0	ESC 28 42
		G1	ESC 2D 46
<i>Hebrew</i>	ISO 2022 IR 138	G0	ESC 28 42
		G1	ESC 2D 48
<i>Latin alphabet No. 5</i>	ISO 2022 IR 148	G0	ESC 28 42
		G1	ESC 2D 4D
<i>Japanese</i>	ISO 2022 IR 13	G0	ESC 28 4A

		G1	ESC 29 49
Thai	ISO 2022 IR 166	G0	ESC 28 42
		G1	ESC 2D 54

Defined Terms for multi-byte character sets with code extensions.			
Name	Defined Term	Code Element	Escape Sequence
Japanese	ISO 2022 IR 87	G0	ESC 24 42
	ISO 2022 IR 159	G0	ESC 24 28 44
Korean	ISO 2022 IR 149	G1	ESC 24 29 43
Chinese Big 5	ISO 2022 B5		ESC 24 29 D7
Guo Biao 2312-80	ISO 2022 GB2312		ESC 24 29 41
Guo Biao 2312-80 with extensions	ISO 2022 GBK		ESC 24 29 41

Defined Terms for multi-byte character sets without code extensions.			
Name	Defined Term	Code Element	Escape Sequence
Unicode in UTF-8	ISO_IR 192		
GB18030	GB18030		

4.8 DICOMScript

A DICOMScript (.DS) describes a single Test Scenario. ACSE Requests & Responses and DIMSE Command / IOD combinations are used to perform a given Test Scenario.

DVT interprets a DICOMScript in order to perform a Test Scenario. See sections on Programming DVT and Advanced Programming for details of DICOMScript contents.

DICOMScripts should be written according to the role played by DVT:

- SCU - write a DICOMScript that plays the SCU role in a given Test Scenario.
- SCP - write a DICOMScript that plays the SCP role in a given Test Scenario.
- FSC – write a DICOMScript that plays the FSC role in a given Test Scenario (creates File-sets).

4.9 DICOMSuperScript

A DICOMSuperScript (.DSS) contains a list of DICOMScript (filenames) that together are used to describe a Test Scenario. DVT executes the DICOMScripts in the order given in the DICOMSuperScript.

The DICOMSuperScript enables the reuse of certain DICOMScripts in various Test Scenarios - e.g., DICOMScript that makes an Association for CT Image Storage.

It is possible to **repeat the DICOMScript** execution a number of times using the

DO n dicomscript

instruction where 'n' is the number of times that *dicomscript* is to be executed.

Example 1:

ASSOC.DS
SC-IMG.DS
REL.DS

Associate using the general ASSOC.DS DICOMScript, Send a Secondary Capture image using the SC-IMG.DS DICOMScript and then release the association using the general REL.DS DICOMScript.

Example 2:

ASSOC.DS
SC-IMG1.DS
REL.DS
ASSOC.DS
SC-IMG2.DS
REL.DS
ASSOC.DS
SC-IMG3.DS
REL.DS

As example one, but send 3 Secondary Capture images (SC-IMG1.DS, SC-IMG2.DS and SC-IMG3.DS), by making an association for each image transfer.

Example 3:

ASSOC.DS
DO 20 SC-IMG.DS
REL.DS

Associate (using ASSOC.DS), repeat Secondary Capture image transfer 20 times and then release the association (using REL.DS).

4.10 VBScript

A DICOM test scenario can be written using the DVTK assemblies in VBScript. The User will need a knowledge of VBScript programming to be able to write DVT VBScripts.

MS HTML Help file (DVTk-API.chm) is available on the DVTk website in developer section describing the DVTK assembly interfaces.

- Use the High Level Interface (HLI) of DVT
- Can contain a complete test scenario
- Can contain a part of a test scenario (re-use)
- Vary from simple to advanced
- Easy to learn when already familiar with Visual Basic Scripting
- Allow advanced Multi-Threaded and Event Driven Conditional Scripting

4.11 Results File

DVT stores the results of each test (either by DICOMScript, DICOMSuperScript, VBScript, Media Validation or Emulation) in a _RES.xml file.

Two types of results files can be generated:

- Detailed Results File – the full validation results are stored in the detailed results file. The *<resultsFilePrefix>* is **Detail**.
- Summary Results File – a summary of the validation results, showing ERROR's and WARNING's, are stored in the summary results file. The *<resultsFilePrefix>* is **Summary**.

For the DICOMScript and DICOMSuperScript, the Result File name is generated from the combination of Session ID (from the Session Properties) and the *<scriptName>*. The following filename is generated:

<resultsFilePrefix>_nnn_<scriptName>_res.xml

where **nnn** is the Session ID.

For Emulation, the Result File name is generated from the combination of Session ID (from the Session Properties), the Result File Index, and the <emulatorType>:

<resultsFilePrefix>_nnn_<emulatorType>_resiini.xml

where **nnn** is the Session ID,
iiii is the file index.

The following <emulatorType> values are possible:

- **Pr_Scp_Em** – Printer SCP Emulator
- **St_Scp_Em** – Storage SCP Emulator
- **St_Scu_Em** – Storage SCU Emulator

For the Media Validation, the Result File name is generated from the combination of Session ID (from the Session Properties) and the <mediaFilename>:

<resultsFilePrefix>_nnn_<mediaFilename>_DCM_res.xml

where **nnn** is the Session ID.

Examples:

Summary_001_11_ds_res.xml	- Summary results for DICOMScript 11.DS used in Script Session 1.
Detail_004_234_dss_res.xml	- Detailed results for DICOMSuperScript 234.DSS used in Script Session 4.
Detail_008_St_Scp_Em_res12.xml	- Detailed results for Results Index 12 used in Storage SCP Emulator Session 8.
Summary_001_DICOMDIR_res.xml	- DICOMDIR File result file.
Summary_001_DICOMDIR_resN.xml	- for each DatasetN of the DICOMDIR File where N=(0..M).

4.11.1 Global Results Files

- Used for filtering results
- Used as input for test reports
- Contains added information on errors
- Contains manually added errors
- You only use this file through the reporting functionality
- Is used on project level not on session level

4.12 Media Storage File

If the *Test Session Properties* **STORAGE-MODE** parameter is set to **as-media**, DVT will store a received Image Dataset (Group 0008 up to and including Group 7FE0) in a file with the extension **.DCM**. The Image Dataset is stored in the .DCM file in the format described in DICOM Standard - part 10. The File Preamble, DICOM Prefix and File Meta Information are added by DVT.

The filename is generated from the Session ID and the media storage file index. The filename used for the media storage is recorded in the corresponding Results File.

The following filenames are generated:

nnnIiii.dcm where **nnn** is the Session ID,
I signifies image information
and **iii** is the file index.

Examples:

1I0123.dcm - Media Storage File 123 created in Test Session 1.

4I0012.dcm - Media Storage File 12 created in Test Session 4.

4.13 Raw Dataset File

If the *Test Session Properties* **STORAGE-MODE** parameter is set to **as-dataset**, DVT will store a received Image Dataset (Group 0008 up to and including Group 7FE0) in a file with the extension **.RAW**. The Dataset content is not parsed and is simply copied into the .RAW file in the Transfer Syntax negotiated.

The filename is generated from the Session ID and the raw dataset file index. The filename used for the raw Dataset of a given image is recorded in the corresponding Results File.

The following filenames are generated:

nnnIiii.raw where **nnn** is the Session ID,
I signifies image information
and **iii** is the file index.

Examples:

1I0123.raw - Raw Image Dataset File 123 created in Test Session 1.

4I0012.raw - Raw Image Dataset File 12 created in Test Session 4.

4.14 Pixel File

If the *Test Session Properties* **STORAGE-MODE** parameter is set to **as-media** or **as-dataset**, when receiving an Image Dataset, DVT stores any OB/OF/OW data in a file with the extension **.PIX**.

The filename is generated from the File Type and the pixel file index.

The filename used for the OB/OF/OW data is recorded in the corresponding Results File.

The following filenames are used / generated:

tttIiii.pix where **ttt** is the File Type,
and **iii** is the file index.

Pixel Filename	
File Type	Description
B08_	Pixel file contains 8-bit, OB data.
B08C	Pixel file contains 8-bit, compressed OB data.
F32L	Pixel file contains 32-bit, little endian, OF data.
F32B	Pixel file contains 32-bit, big endian, OF data.
W08L	Pixel file contains 8-bit, little endian OW data.
W08B	Pixel file contains 8-bit, big endian OW data.
W16L	Pixel file contains 16-bit, little endian OW data.
W16B	Pixel file contains 16-bit, big endian OW data.

The File Type, forms part of the pixel filename, and is used by DVT to determine how the .PIX file should be read when transmitting the contents according to the agreed Transfer Syntax. DVT cannot compress or uncompress data, so the data in the file must match the agreed Transfer Syntax. No warning is given if the data does not match the Transfer

Syntax. When receiving OB/OF/OW data, DVT does not transform the OB/OF/OW data from the agreed Transfer Syntax, but will generate a .PIX File Type matching the way in which the OB/OF/OW data was received.

DVT also writes a short description of the OB/OF/OW data as a header in the Pixel File. The header has the following format:

Pixel File Header		
Byte Offset	Value	Description
0..1	0x002A or 0x2A00	File endian-ness. Value 002A indicates that the remaining header and OB/OF/OW data is stored in Big Endian format. Value 2A00 indicates that the remaining header and OB/OF/OW data is stored in Little Endian format.
2..3	0x0000	Version Tag = 0
4..7	0x00000002	Version Tag Length = 2
8..9	0x0002	Version Value = 3 (current DVT version)
10..11	0x0001	VR Tag = 1
12..15	0x00000002	VR Length = 2
16..17	0x4242, 0x4646 or 0x5757	VR Value. 0x4242 = "BB" which indicates that the OB/OF/OW data is of DICOM VR OB. 0x4646 = "FF" which indicates that the OB/OF/OW data is of DICOM VR OF. 0x5757 = "WW" which indicates that the OB/OF/OW data is of DICOM VR OW.
18..19	0x0002	BitsAllocated Tag = 2
20..23	0x00000002	BitsAllocated Length = 2
24..25	0x0008, 0x0010 or 0x0020	BitsAllocated Value. 0x0008 indicates 8-bit OB/OW data. 0x0010 indicates 16-bit OB/OW data. 0x0020 indicates 32-bit OF data.
26..27	0x0003	TransferSyntax Tag = 3
28..31	0x00000041	TransferSyntax Length = 65
32..96	UID	TransferSyntax Value – null terminated.
97..98	0x7FE0	OBOFOWDataLength Tag
99..100	0x00000004	OBOFOWDataLength Length
101..104	Nnnnnnnn	OBOFOWDataLength Value. Nnnnnnnn indicates the length of the OB/OF/OW data in bytes from the last byte of this OBOFOWDataLength Value.

DVT can read Pixel Files without the Pixel File Header, in which case the filename is parsed in order to determine the OB/OF/OW data details.

When a Pixel File contains Compressed (Encapsulated) data, it is up to the User to ensure that the file contains a correct Basic Offset Table. DVT will simply read the Basic Offset Table as part of the OB/OF/OW data during encoding. DVT will write any Basic Offset Table received into the Pixel File at the start position of the OB/OF/OW data. DVT will validate the Basic Offset Table contents.

Example Filenames:

- B08_0123.pix - Image Pixel File 123 contains 8-bit OB data.
- B08C0012.pix - Image Pixel File 12 contains 8-bit compressed OB data.
- W16L0003.pix - Image Pixel File 3 contains 16-bit little endian OW data.

4.15 File Index File

The *storage.idx* is a text file used to maintain the file indices for the various file types generated by DVT. The file consists of a single line containing the next .DCM index, followed by the next .RAW index, then the next .PIX index and finally the .RES index (used for emulator results). A SPACE character separates the values.

The User can edit the file contents to start the file indexing off from a required number. The storage.idx file can be deleted to cause DVT to start indexing each file type from 1.

5. Graphical User Interface - GUI

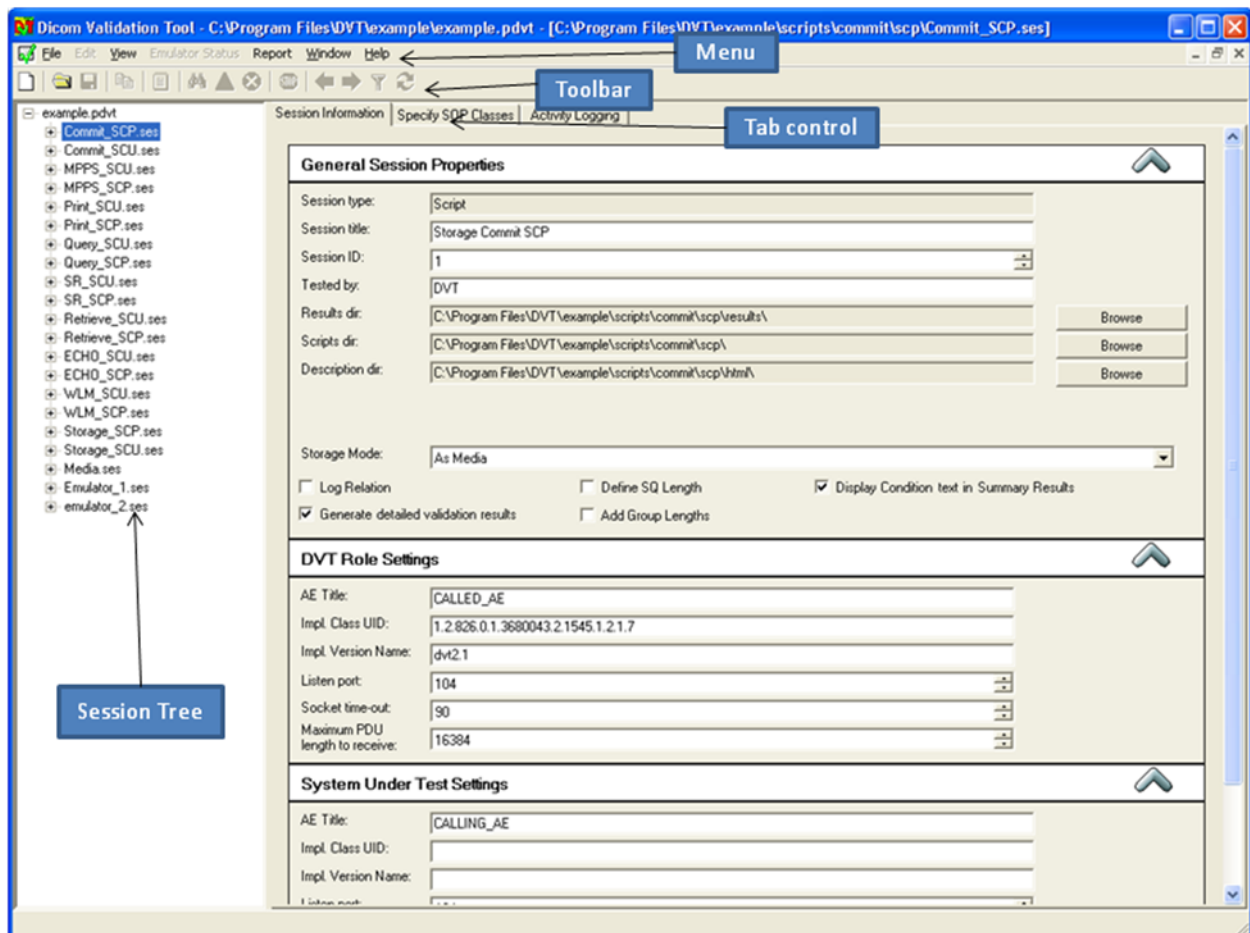
A GUI for DVT is available to be able to work in a user-friendly way with the Sessions, which are part of a Project.

Three separate ways exist by which to start working with the GUI:

- Start working with a Project or Session by supplying the full filename as argument to Dvt.exe.
- Start DVT and load an existing Project or Session (a new Project will be automatically created for this Session).
- Start DVT and create a new Project.

The GUI consists of the following parts (see picture below):

- Menu
- Toolbar
- One or more project views (only when a project is loaded), each project view consisting of
 - Session Tree
 - Tab control, the tab control consisting of
 - One or more tabs



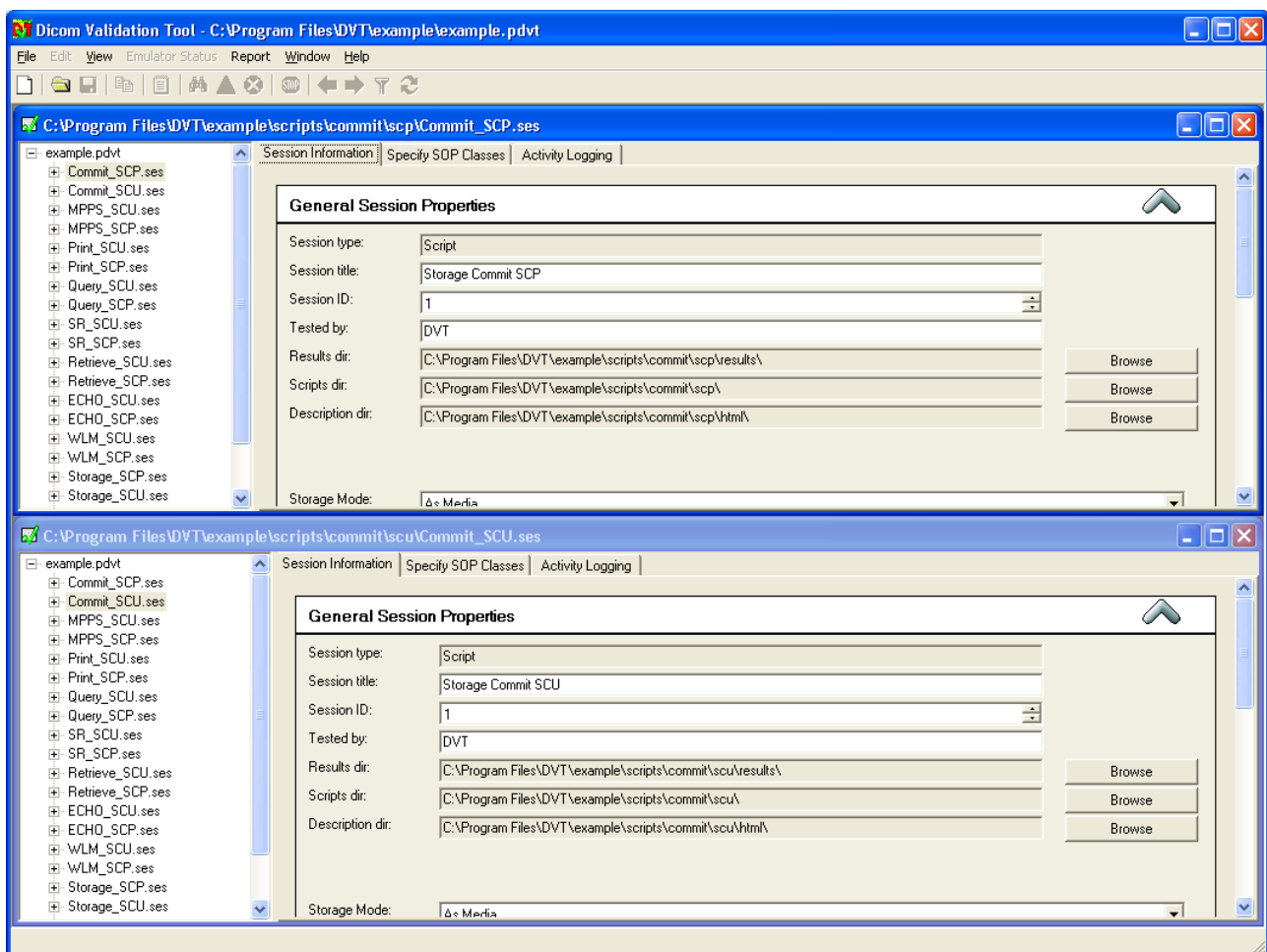
GUI with one Project view

Depending on which node is selected in the Session Tree,

- Specific menu (items) will be enabled.
- Specific toolbar buttons will be enabled.
- Specific tabs will be visible.

See also the following sections for a detailed description of the relation between the Session Tree and other parts of a Project view.

When a Project is loaded, one or more Project views (more Project views may be added using the Window menu) may be present in the GUI (see picture below for a GUI with two project views). It is important to note that all Project views are visualizing the same Project. So when changing something in one Project view, this change is also visible (if the same changed object is selected in the Session tree) in the other Project views.



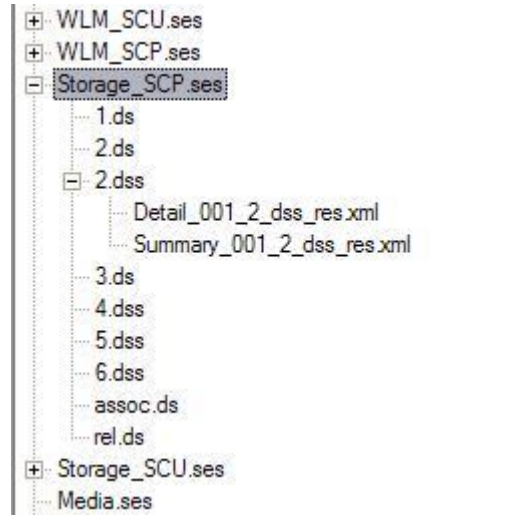
GUI with two Project views

Only one Project View is the active one: the one with the non-ghosted title bar. The specific menu items and toolbar buttons that will be enabled only depend on the selected node in the Session tree of the active Project view.

In the rest of the sections, when referring to the Session tree, tab control, etc, the Session tree, tab control of the active Project view is meant.

5.1 Session Tree

The Session Tree visualizes different kind of objects, like e.g. a Session, a Script, etc.



Part of a Session Tree

The following table gives an overview of how objects are visualized in the Session Tree and how they are stored:

Object	Part of	Visualized by	Stored in
Project	-	Complete Session tree.	Project file (*.pdvt)
Script Session	Project	Script Session node.	Session file (*.ses) of type Script.
Emulator Session	Project	Emulator Session node.	Session file (*.ses) of type Emulator.
Media Session	Project	Media Session node.	Session file (*.ses) of type Media.
Script	Session	Script node.	Script file (*.ds, *.dss or *.vbs).
Results	Script, Emulator or Results collection	Results node.	Results file (*.xml)
Storage SCP Emulator	Emulator Session	Storage SCP Emulator node.	-
Storage SCU Emulator	Emulator Session	Storage SCU Emulator node.	-
Print SCP Emulator	Emulator Session	Print SCP Emulator node.	-
Results collection	Media Session	Results collection node.	-

At most one node may be selected in the Session Tree. For the rest of this chapter, the following convention will be used: when a node in the Session Tree is selected, the object visualized by the selected node will be called the selected object. So e.g. “selected Session” is the Session visualized by the selected Session node in the Session tree.

Each type of Session is differently visualized in the Session Tree. The level of the (sub) nodes for each type of Session node in the Session Tree is now summarized:

1) Script Session

- 2) Script (zero or more scripts that are stored in the Script directory of the Session)
- 3) Results (zero or more Results for each Script)

1) Emulator Session

- 2) Storage SCP Emulator
- 3) Results (zero or more Results)

2) Storage SCU Emulator

3) Results (zero or more Results)

2) Print SCP Emulator

3) Results (zero or more Results)

1) Media Session

2) Results collection

3) Results (one or more Results)

5.2 Tab Control

The Tab Control contains several tabs. Depending on which node is selected in the Session tree, a subset of these tabs will only be visible. In the example below, a Session node is selected and the tabs shown will be visible.

The screenshot displays the 'Session Information' tab in the DVT application. It features three main sections for configuring a session:

- General Session Properties:** Includes fields for Session type (Script), Session title (Storage Commit SCP), Session ID (1), Tested by (DVT), Results dir, Scripts dir, and Description dir, each with a corresponding 'Browse' button. It also has a Storage Mode dropdown set to 'As Media' and several checkboxes for logging and validation options.
- DVT Role Settings:** Includes fields for AE Title (CALLED_AE), Impl. Class UID, Impl. Version Name (dvt2.1), Listen port (104), Socket time-out (90), and Maximum PDU length to receive (16384).
- System Under Test Settings:** Includes fields for AE Title (CALLING_AE), Impl. Class UID, Impl. Version Name, Listen port (104), Remote TCP/IP address (localhost), and Maximum PDU length to receive (16384).

At the bottom, there is a checkbox for 'Security Settings'.

The tab control

In the rest of this section, all different tabs will be discussed.

5.2.1 Session Information Tab

The Session Information tab is divided into several panels. Each panel will be discussed separate below.

See the previous section for a screenshot of the Session Information tab.

5.2.1.1 General Session Properties Panel

General Session Properties

Session type: Script

Session title: Storage Commit SCP

Session ID: 1

Tested by: DVT

Results dir: C:\Program Files\DVT\example\scripts\commit\scp\results\ Browse

Scripts dir: C:\Program Files\DVT\example\scripts\commit\scp\ Browse

Description dir: C:\Program Files\DVT\example\scripts\commit\scp\html\ Browse

Storage Mode: As Media

☐ Log Relation ☐ Define SQ Length ☒ Display Condition text in Summary Results

☒ Generate detailed validation results ☐ Add Group Lengths

The “General Session Properties” panel

- **Session type (cannot be changed)** – see SUT Test Session Property SESSION-TYPE.
- **Session title** – see SUT Test Session Property SESSION-TITLE.
- **Session ID** – see SUT Test Session Property SESSION-ID.
- **Tested by** – see SUT Test Session Property TESTED-BY.
- **Results dir** – see Test Session Property RESULTS-ROOT.
- **Scripts dir (only present for a Script Session)** – see Test Session Property DICOMSCRIPT-ROOT.
- **Description dir (only present for a Script Session)** – see Test Session Property DESCRIPTION-DIRECTORY.
- **Storage Mode** – see Test Session Property STORAGE-MODE.
- **Log Relation** – see Test Session Property LOG-RELATION.
- **Generate detailed validation results** – see Test Session Property DETAILED-VALIDATION-RESULTS.
Note: Summary validation Results will always be generated.

5.2.1.2 DVT Role Settings Panel

This panel is only visible when the Session Information tab is displaying information for a Script Session or Emulator Session.

DVT Role Settings

AE Title: CALLED_AE

Impl. Class UID: 100.118.116.2004.2.0

Impl. Version Name: dvt2.0

Listen port: 104

Socket time-out: 90

Maximum PDU length to receive: 16384

The “DVT Role Settings” panel

- **AE Title** – see DVT ACSE Property DVT-AE-TITLE.
- **Impl. Class UID** – see DVT ACSE Property DVT-IMPLEMENTATION-CLASS-UID.

- **Impl. Version Name**— see DVT ACSE Property DVT-IMPLEMENTATION-CLASS-NAME.

***NOTE:** The Implementation Version Name is an optional field in DICOM. When DVT should not send this value, leave this entry blank.*

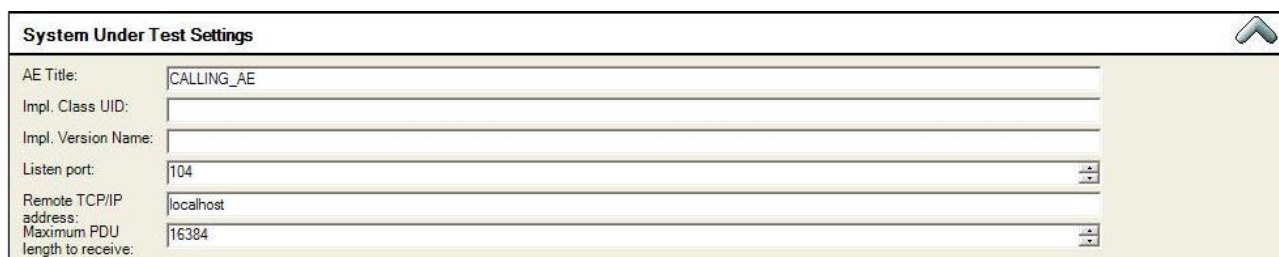
- **Listen port** — see Socket Property DVT-PORT.

***NOTE:** The default port for DICOM applications, when using no secure connections, is 104. The default port for DICOM applications, when using secure connections, is 2762.*

- **Socket time-out**— see Socket Property DVT-SOCKET-TIMEOUT.
- **Maximum PDU length to receive**— see DVT ACSE Property DVT-MAXIMUM-LENGTH-RECEIVED.

5.2.1.3 System Under Test Settings Panel

This panel is only visible when the Session Information tab is displaying information for a Script Session or Emulator Session.



The “System Under Test Settings” panel

- **AE Title**— see SUT ACSE Property SUT-AE-TITLE.
- **Impl. Class UID** — see SUT ACSE Property SUT-IMPLEMENTATION-CLASS-UID.
- **Impl. Version Name**— see SUT ACSE Property SUT-IMPLEMENTATION-VERSION-NAME.

***NOTE:** The Implementation Version Name is an optional field in DICOM. When the SUT does not send this value, leave this entry blank.*

- **Listen port**— see Socket Property SUT-PORT.

***NOTE:** The default port for DICOM applications, when using no secure connections, is 104. The default port for DICOM applications, when using secure connections, is 2762.*

- **Remote TCP/IP address**— see Socket Property SUT-HOSTNAME.
- **Maximum PDU length to receive**— see SUT ACSE Property SUT-MAXIMUM-LENGTH-RECEIVED.

5.2.1.4 Security Settings Panel

This panel is only visible when the Session Information tab is displaying information for a Script Session or Emulator Session.

Only when the checkbox “Secure Settings” is checked, secure sockets will be used for communication between DVT and the SUT. When it is not checked, standard TCP/IP is used.



The “Security Settings” panel

The settings in the following sub sections will only be applied when secure sockets are used for communication between DVT and the SUT.

Note: From version 2.6.9 a new version of Openssl is introduced, there is only one option at a time selectable for authentication, key exchange, data integrity and encryption. The checkboxes are replaced by radiobuttons. If a sessionfile with more options for one category is opened default options will be loaded.

5.2.1.4.1 General

- **Check remote certificates.** Only when this checkbox is checked, remote private keys that are received will be checked against the DVT file containing SUT public keys. If the received private key is invalid, the connection will be closed.

Regardless whether the checking of remote private keys is enabled or not, DVT will do the following:

- If the SUT initiates a secure socket connection, DVT will request a private key from the SUT. If no private key is returned, the connection will be closed.
- If DVT initiates a secure socket connection, the connection will be closed if the SUT does not return a private key.

See Socket Property – CHECK-REMOTE-CERTIFICATE.

- **Cache secure sessions.** Only when this checkbox is enabled, caching of secure socket Sessions will be enabled.

See Socket Property – CACHE-TLS-SESSIONS.

5.2.1.4.2 Version

At least one of the four TLS version need to be selected(TLSv1, TLSv1.1, TLSv1.2, TLSv1.3) for max and min TLS version. Max TLS version cannot be smaller than min tls version.

See Socket Property MAX-TLS-VERSION, MIN-TLS-VERSION.

5.2.1.4.3 Authentication

At least and only one of the following two authentications must be enabled.

- **RSA.** Only when this checkbox is checked, the authentication RSA will be enabled.
- **DSA.** Only when this checkbox is checked, the authentication DSA will be enabled.

See Socket Property CIPHER-LIST.

5.2.1.4.4 Key Exchange

At least and only one of the following two key exchanges must be enabled.

- **RSA.** Only when this checkbox is checked, the key exchange RSA will be enabled.
- **DH.** Only when this checkbox is checked, the key exchange DH will be enabled.

See Socket Property CIPHER-LIST.

5.2.1.4.5 Data Integrity

At least and only one of the following two data integrities must be enabled.

- **SHA.** Only when this checkbox is checked, the data integrity SHA will be enabled.
- **MD5.** Only when this checkbox is checked, the data integrity MD5 will be enabled.

See Socket Property CIPHER-LIST.

5.2.1.4.6 Encryption

At least and only one of the following four encryptions must be enabled.

- **None.** Only when this checkbox is checked, no encryption will be enabled.
- **Triple DES.** Only when this checkbox is checked, the encryption Triple DES will be enabled.
- **AES 128-bit.** Only when this checkbox is checked, the encryption AES 128-bit will be enabled.
- **AES 256-bit.** Only when this checkbox is checked, the encryption AES 256-bit will be enabled.

See Socket Property CIPHER-LIST.

5.2.1.4.7 Keys

- **File containing SUT Public Keys (certificates), the Browse button.** A file browser dialog is presented by which the file containing SUT public keys may be selected.
- **File containing SUT Public Keys (certificates), the Edit button.** A dialog is presented (see below for a screenshot), in which the SUT public keys are displayed. In the same dialog, public keys may be added and removed.

When the “Import Key” button is pressed, a public key may be imported. See the table below for the different kind of key formats supported.

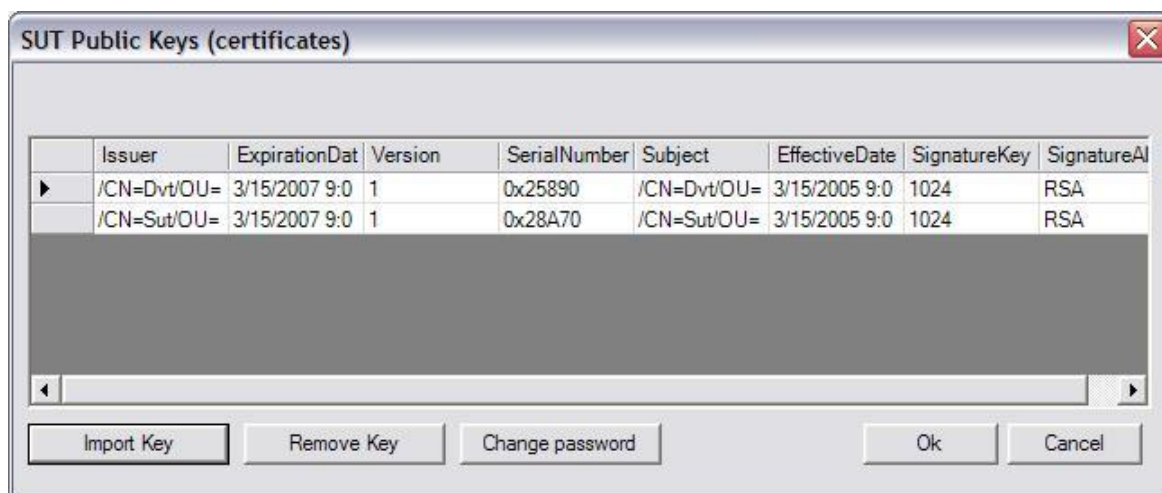
When the “Remove Key” button is pressed, the key selected will be removed.

The “Change Password” button displays a dialog box that allows the User to change the password that is used to encrypt the public keys in the public keys file. If the “use default password” checkbox is checked, the internal DVT password is used.

Any changes made in the dialog are not saved until the OK button is pressed. If the Cancel button is pressed, all changes are discarded.

Format	File Extensions
PEM in Base64	.pem, .cer
DER (binary PEM)	.cer
PKCS #7	.p7b, .p7c
PKCS #12	.p12, .pfx

The different key formats supported when importing a key.



The “SUT Public Keys (certificates)” dialog

- **File containing DVT Private Keys (credentials), the Browse button.** A file browser dialog is presented by which the file containing DVT private keys may be selected.
- **File containing DVT Private Keys (credentials), the Edit button.** A dialog is presented, in which the DVT private keys are displayed. In the same dialog, private keys may be added and removed.

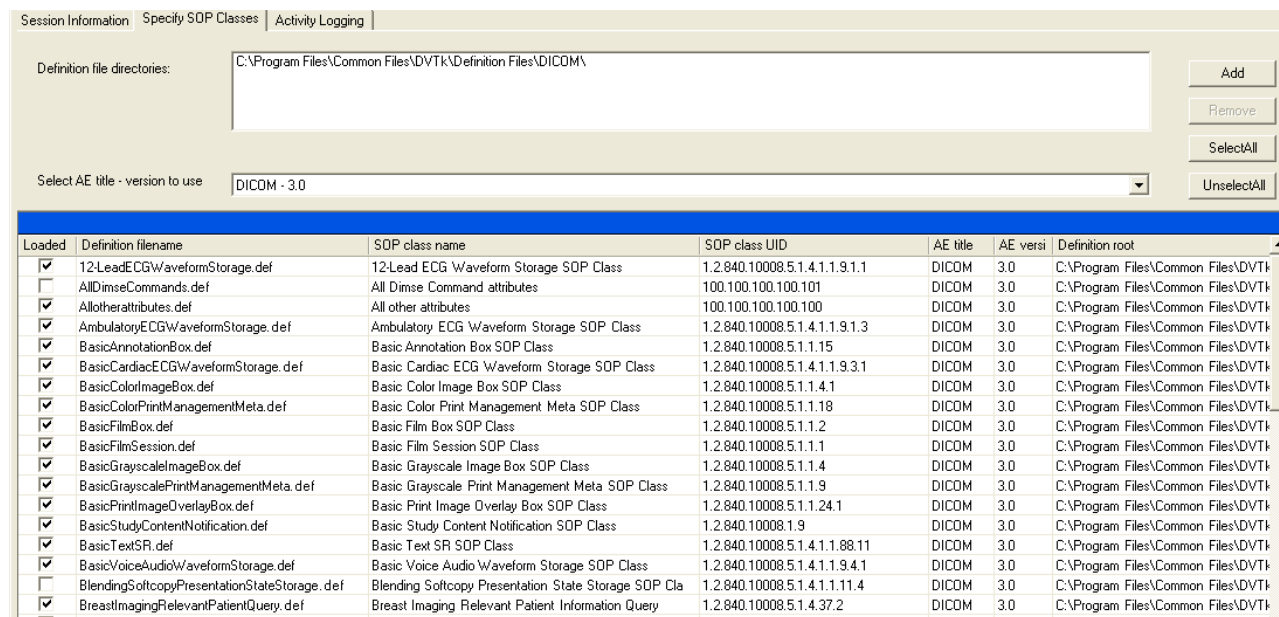
See “File containing SUT Public Keys (certificates), the Edit button” for more information (the “DVT Private Keys (credentials)” dialog is similar to the “SUT Public Keys (certificates)” dialog).

- **Create Public/Private key pair, the Create Button.** A wizard is presented, by which a public/private key pair may be generated.

See Socket Properties CREDENTIALS-FILENAME and CERTIFICATE-FILENAME.

5.2.2 Specify SOP Classes Tab

The Specify SOP Classes tab shows which SOP classes are loaded in the selected Session.



The “Specify SOP Classes” tab

All definition files that are present in the directories specified in the “Definition file directories” listbox are shown. Each definition file contains information about a specific SOP class. Definition file directories may be added or removed by

using the Add and Remove button. Removing a definition file directory is only allowed if more than one definition file directory is present. By enabling or disabling the “Loaded” checkbox, the user determines which SOP classes are loaded. Finally, the “Select AE title – version to use” combobox determines which of the loaded SOP classes are actually used during validation.

See properties DEFINITION-ROOT, DEFINITION, APPLICATION-ENTITY-NAME and APPLICATION-ENTITY-VERSION.

5.2.3 Activity Logging Tab

The activity logging tab shows detailed progress information during script execution, emulator execution and media validation.

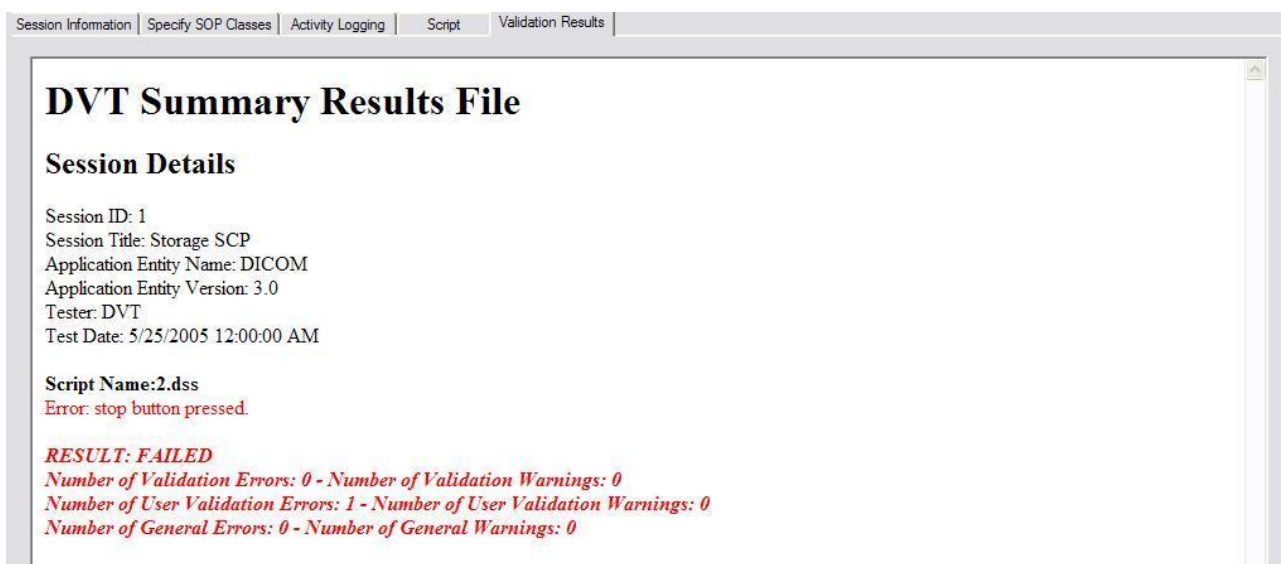


The “Activity Logging” tab

Note that after execution or validation, the Activity Logging tab shows information about the last execution or validation. The content of the tab is not related to e.g. a selected Result.

5.2.4 Validation Results Tab

The validation results tab shows the content of the selected Result.



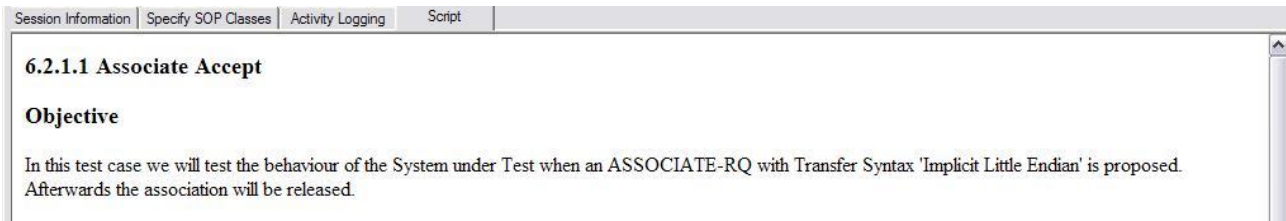
The “Validation Results” tab

5.2.5 Script Tab

The Script tab shows a script description or script content (if a description does not exist) for the selected Script.



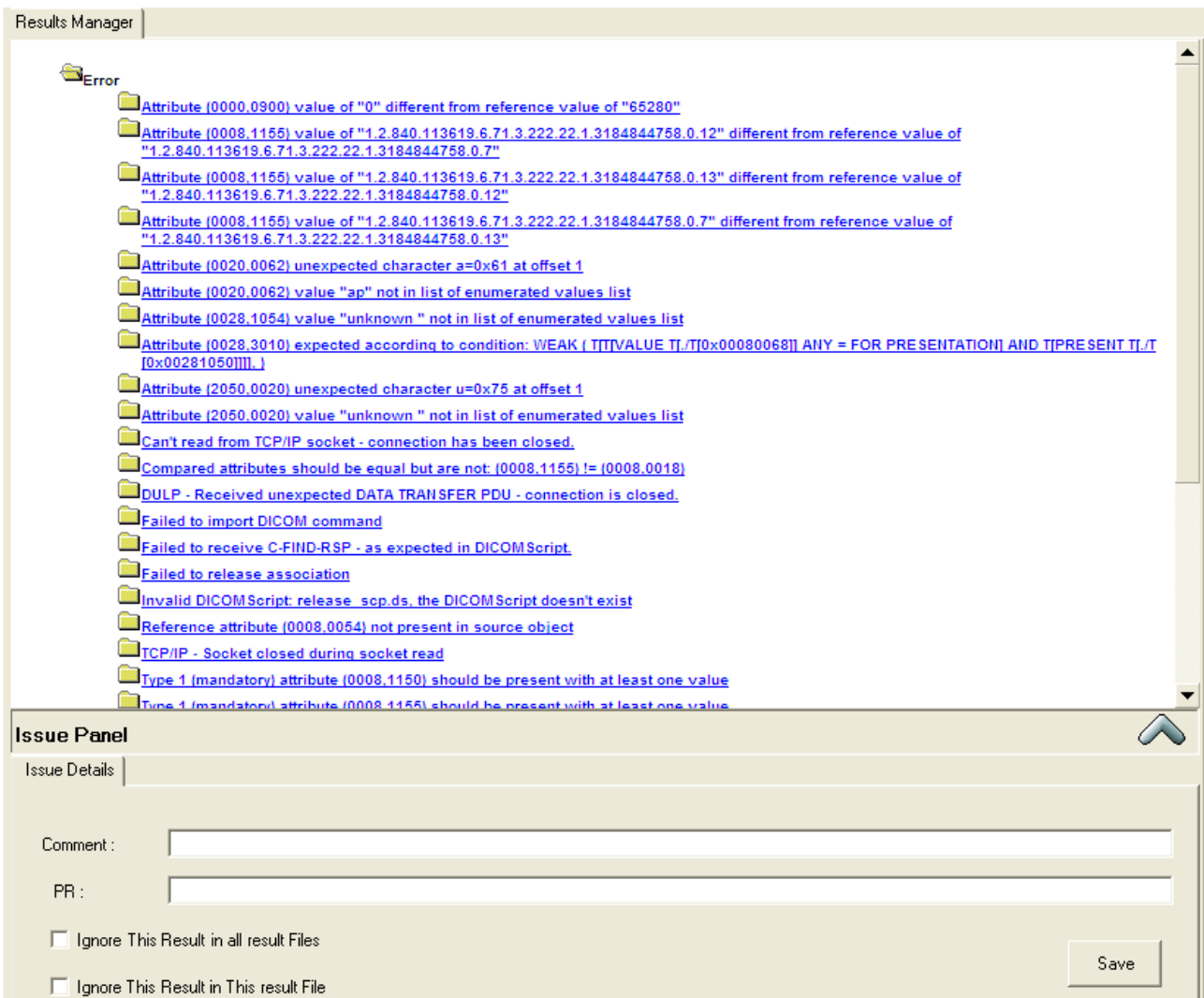
The “Script” tab with Dicom Super Script content



The “Script” tab with a HTML description

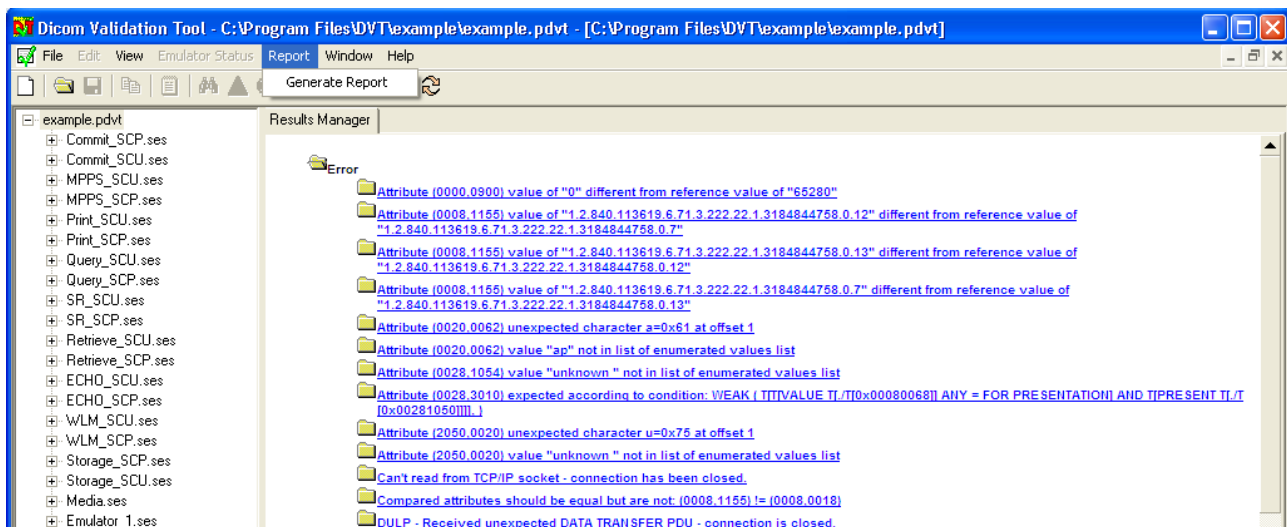
See property DESCRIPTION-DIRECTORY.

5.2.6 Results Manager Tab



This tab provides the new reporting functionality in DVT GUI. It provides following information:

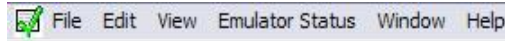
- Overview of problems over different scripts/sessions:
 - How to keep an overview of so many results?
 - How to handle duplicates of Errors and Warnings?
- Managing Problems
 - How to 'hide' non-relevant Errors and Warnings in logging?
 - How to add a problem to a result file?
 - How to relate Errors and Warnings to a Problem Reports?
- Generation of HTML consolidated report
 - How to put result and interpretation easily in a report?



Generate Report option in menu

5.3 Main Menu Bar

This section described the functionality behind all menu items of the main menu bar.



The main menu bar

5.3.1 File Menu

The section describes the functionality behind the different File menu items.



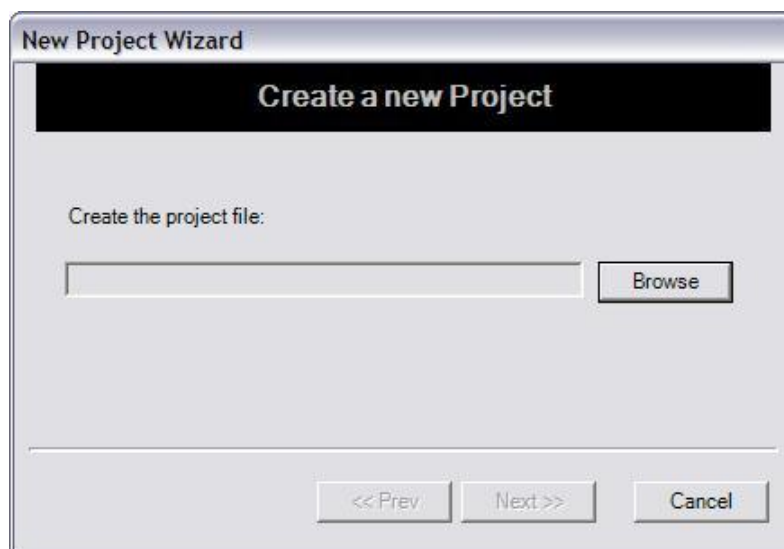
The “File” menu

5.3.1.1 New...

Create a new Project using a wizard (the menu item is only enabled when no Session is executing).

If a Project is already loaded and unsaved changes exist in the Project and/or Session(s), a dialog is presented in which the user can decide to save (some of) the changes, not save any changes or cancel the creation of the new Project.

The created Project will automatically be written to the Project file, as specified in the wizard.



First screen of the “New Project Wizard” in which to specify the Project file name

5.3.1.2 Open...

Open an existing Project or Session (the menu item is only enabled when no Session is executing).

If a Project is already loaded and unsaved changes exist in the Project and/or Session(s), a dialog is presented in which the user can decide to save (some of) the changes, not save any changes or cancel the opening of the Project or Session.

A file browser is presented, by which the user can load a Project file or Session file.

NOTE: When a Session file is loaded, a new Project will be created containing the loaded Session. The user has to explicitly save this automatically created Project to file if needed.

5.3.1.3 Save...

Save unsaved changes to the Project file and/or Session file(s) (the menu item is only enabled when no Session is executing).

A dialog is presented by which the user can decide what (changed Project and/or changed Session(s)) needs to be written to file.

5.3.1.4 Project \ Add Existing Session(s)...

Add an existing Session to the Project (the menu item is only enabled when a Project is loaded).

A file browser is presented, by which the user can load a Session file.

NOTE: The user has to explicitly save the changed Project to file (the Session is added to it so it has changed) if needed.

5.3.1.5 Project \ Add New Session...

Create a new Session and add it to the Project using a wizard (the menu item is only enabled when a Project is loaded).

The created Session will automatically be written to the specified Session file, as specified in the wizard.

NOTE: The user has to explicitly save the changed Project to file (the Session is added to it so it has changed) if needed.

5.3.1.6 Project \ Save (Project file only)

Save the Project to its Project file.

NOTE: Any changed Session will not be written to file.

5.3.1.7 Project \ Save As (Project file only)...

Save the Project to a specified Project file.

A file browser is presented, by which the user can specify a new Project file. The Project will be saved to this specified Project file. Future saves will also be saved in this specified Project file. The contents of the current Project file will not change.

NOTE: Any changed Session will not be written to file.

5.3.1.8 Session \ Remove Session <Session file name> from Project

Remove the selected Session from the Project (the menu item is only enabled when a Session is selected which is not executing).

A warning dialog is presented in which the user has to confirm the removal of the Session.

If unsaved changes exist in the selected Session, a dialog is presented in which the user can decide to save the changes, not save any changes or cancel the removing of the Session.

NOTE: While the Session is removed from the Project, the Session file is not removed.

5.3.1.9 Session \ Save Session <Session file name>

Save the selected Session to its Session file (the menu item is only enabled when a Session is selected which is not executing).

5.3.1.10 Session \ Save As...

Save the selected Session to a specified Session file (the menu item is only enabled when a Session is selected which is not executing).

A file browser is presented, by which the user can specify a new Session file. The Session will be saved to this specified Session file. Future saves will also be saved in this specified Session file. The contents of the current Session file will not change.

5.3.1.11 Exit

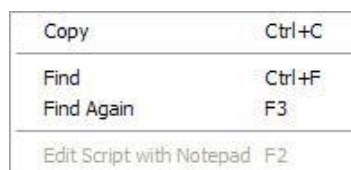
Exit the application.

If execution is going on, a dialog is presented by which the user can decide whether to stop the execution or cancel the exit of the application.

If unsaved changes exist in the Project and/or Session(s), a dialog is presented in which the user can decide to save (some of) the changes, not save any changes or cancel the exit of the application.

5.3.2 Edit Menu

This section describes the functionality behind the different Edit menu items.



The “Edit” menu

5.3.2.1 Copy

Copy the selected text to the ClipBoard (the menu item is only enabled when the tab control shows the Script tab, the Activity Logging tab or the Validation Results tab).

When no text is selected, an empty string will be copied to the clipboard.

5.3.2.2 Find

Find a text string in the selected validation results (the menu item is only enabled when the tab control shows the Validation Results tab).



The “Find” dialog

The last text string that is entered in the “Find what” text box is also used for the “Find again” menu item.

5.3.2.3 Find Again

Find again some text string in the Validation Results tab (the menu item is only enabled when the tab control shows the Validation Results tab and a non-empty text string was specified the last time the Find dialog was used).

The last text string that was entered in the text box of the Find dialog is used for finding the next occurrence in the Validation Results tab.

5.3.2.4 Edit Script with Notepad...

Edit the selected script with notepad (the menu item is only enabled when a script is selected).

When the changed script is saved in Notepad (maybe to another file) this is not automatically reflected in the GUI, because saving of the file is done outside DVT.

When the script contents has changed, and the new contents needs to be visible in the GUI, do one of the following:

- Use the menu item “View menu\Refresh Session Tree”.
- Select another Session Tree node and reselect the script.

When the script has been saved to a new file (and this new file is present in the script directory of one of the Session) and this new script needs to be visible in the User Interface, do the following:

1. Use the menu item “View menu\Refresh Session Tree”.
2. In the Session tree, expand the Script Session containing the new script.

The shortcut key for this menu item is the **F2** key.

5.3.3 View Menu

This section describes the functionality behind the View menu item:



The “View” menu

5.3.3.1 Ask for Backup

Enable or disable the “Ask for Backup” User setting.

Before executing a Script or Emulator, or validating Media file(s), DVT checks if old Results exist that will be removed before the actual execution or validation takes place. Only when this menu item is checked, a dialog is presented in which the user can decide to copy the Results file(s) to backup file(s), not copy the Results file(s) or cancel the execution or validation.

5.3.3.2 Show Dicom Scripts

Enable or disable the “Show Dicom Scripts” User setting.

Only when this menu item is checked, the Dicom Scripts will be visible in the Session Tree.

The Session Tree is completely refreshed after changing the checked state of this menu item.

5.3.3.3 Show Dicom Super Scripts

Enable or disable the “Show Dicom Super Scripts” User setting.

Only when this menu item is checked, the Dicom Super Scripts will be visible in the Session Tree.

The Session Tree is completely refreshed after changing the checked state of this menu item.

5.3.3.4 Show Visual Basic Scripts

Enable or disable the “Show Visual Basic Scripts” User setting.

Only when this menu item is checked, the Visual Basic Scripts will be visible in the Session Tree.

The Session Tree is completely refreshed after changing the checked state of this menu item.

5.3.3.5 Show Empty Script Sessions

Enable or disable the “Show Empty Script Sessions” User setting.

When this menu item is checked, all Sessions will be visible in the Session Tree. When this menu item is not checked, only Script Sessions containing visible Scripts will be visible. Emulator and Media Sessions will always be visible, regardless of the checked state of this menu item.

5.3.3.6 Refresh Session Tree

Refresh the Session Tree.

The Project file, Session file(s) and files\directories referred to by the Session file(s) are re-inspecting after which the Session Tree is reconstructed again from scratch.

This menu item is typically used when changes have been made, outside DVT, to files that DVT uses. E.g. when Results files have been removed from the Results directory of one of the Project Sessions outside DVT, use the Refresh menu item to reflect this change in the GUI.

The shortcut key for this menu item is the **F5** key.

5.3.4 Emulator Status Menu

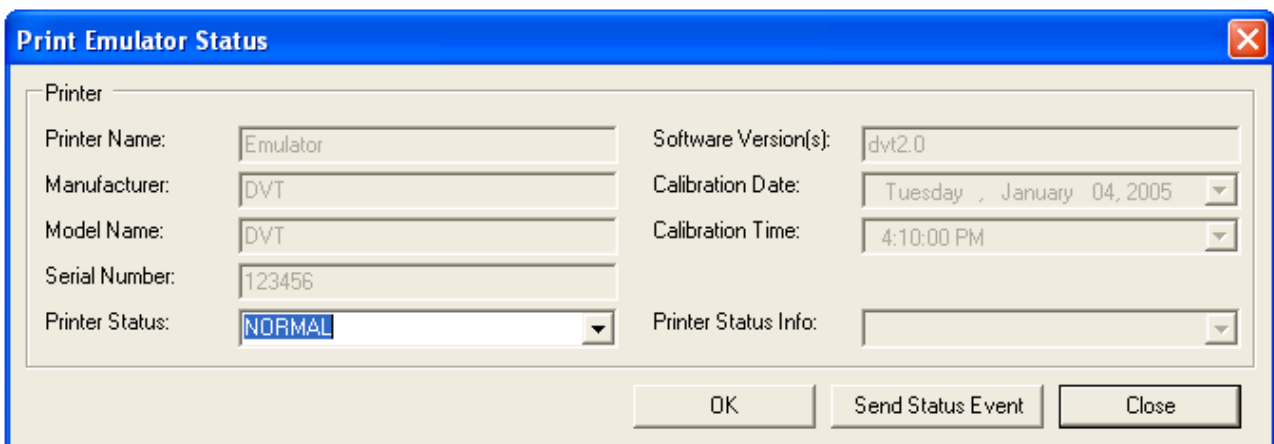
This section describes the functionality behind the Emulator Status menu item.

Status Print SCP Emulator

The “Emulator Status” menu

5.3.4.1 Status Print SCP Emulator

Present a Print Emulator Status dialog (the menu item is only enabled when the Print SCP Emulator is executing).



The “Print Emulator Status” dialog

Pressing the “OK” button will do the following:

- Change the printer status within the emulator.
- When the Print SCP emulator receives an N-GET request, it will use the new printer status in the response.

Pressing the “Send Status Event” button will do the following:

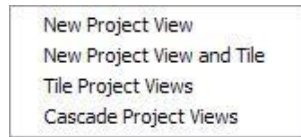
- Issue an N-EVENT if an association is open with the print host.

Pressing the “Close” button will close the dialog without change.

The “Printer Status Info” combo box values are taken from the loaded Print Definition File - Printer SOP Definition - defined terms.

5.3.5 Window Menu

This section describes the functionality behind the different Window menu items.



The “Window” menu

5.3.5.1 New Project View

Open an extra Project view.

5.3.5.2 New Project View and Tile

Open an extra Project view and tile the Project view alongside each other.

5.3.5.3 Tile Project Views

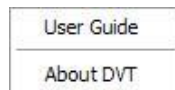
Tile the Project views alongside each other.

5.3.5.4 Cascade Project Views

Cascade the Project views one behind the other.

5.3.6 Help Menu

This section describes the functionality behind the Help menu item.



The “Help” menu

5.3.6.1 User Guide

Show the DVT User Guide.

5.3.6.2 About DVT

Present a dialog with information about the current DVT software release.

5.4 ToolBar

The toolbar contains often-used functionality. Depending on the context (Project loaded or not, which Session Tree node is selected, etc), a toolbar button will be enabled or disabled.



The toolbar

The rest of this section describes the functionality behind each toolbar buttons.

5.4.1 New

Create a new Project using a wizard (the toolbar button is only enabled when no Session is executing).

See 5.3.1.1 (Menu item “New...”) for more information.

5.4.2 Open

Open an existing Project or Session (the toolbar button is only enabled when no Session is executing).

See 5.3.1.2 (Menu item “Open...”) for more information.

5.4.3 Save

Save unsaved changes to the Project file and/or Session file(s) (the toolbar button is only enabled when no Session is executing).

See 5.3.1.3 (Menu item “Save...”) for more information.

5.4.4 Copy

Copy the selected text to the ClipBoard (the toolbar button is only enabled when the tab control shows the Script tab, the Activity Logging tab or the Validation Results tab).

See 5.3.2.1 (Menu item “Copy”) for more information.

5.4.5 Edit Script with Notepad

Edit the selected script with notepad (the toolbar button is only enabled when a script is selected).

See 5.3.2.4 (Menu item “Edit Script with Notepad...”) for more information.

5.4.6 Find

Find a text string in the selected Results (the toolbar button is only enabled when the tab control shows the Validation Results tab).

See 5.3.2.2 (Menu item “Find”) for more information.

5.4.7 Find Next Warning

Find the next warning in the selected Results (the toolbar button is only enabled when the Results shows the Validation Results tab and warning(s) exist in the validation results shown).

5.4.8 Find Next Error

Find the next error in the selected Results (the toolbar button is only enabled when the tab control shows the Validation Results tab and error(s) exist in the validation results shown).

5.4.9 Stop

Stop execution of the Script or Emulator (the toolbar button is only enabled when the selected Session is executing)

5.4.10 **Navigate Back**

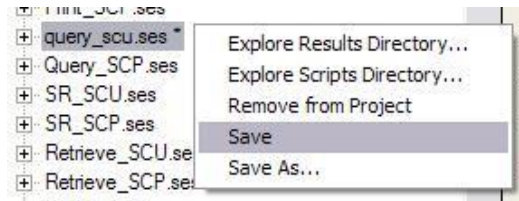
Navigate back to a previously shown Results (the toolbar button is only enabled when the tab control shows the Validation Results tab and it is possible to navigate back).

5.4.11 **Navigate Forward**

Navigate forward to a previously shown Results (the toolbar button is only enabled when the tab control shows the Validation Results tab and it is possible to navigate forward).

5.5 Context menu of the Session Tree

When the mouse pointer is on top of the Session Tree and the right mouse button is clicked, a context menu is presented. Depending on which type of Session Tree node the mouse pointer is on top of, a different context menu is shown. This section describes all context menu items that are present for each type of Session Tree node.



Example of a Session Tree context menu

5.5.1 Session node

This section describes the common context menu items that are present for a Script Session node, Emulator Session node and Media Session node.

5.5.1.1 Explore Results Directory...

Opens Windows Explorer, displaying the Results directory of the Session.

5.5.1.2 Remove From Project

Remove the selected Session from the Project (the context menu item is only enabled when the Session is not executing).

See 5.3.1.8 (Menu item “Session \ Remove Session <Session file name> from Project”) for more information.

5.5.1.3 Save

Save unsaved changes to the Session file (the context menu item is only enabled when unsaved changes exist for this Session).

5.5.1.4 Save As...

Save the selected Session to a specified Session file (the context menu item is only enabled the Session is not executing).

See 5.3.1.9 (Menu item “Session \ Save Session <Session file name>”) for more information.

5.5.2 Script Session node

This section describes the context menu items that are present for a Script Session Node that are not already discussed in 5.5.1 (Session node).

5.5.2.1 Explore Scripts Directory...

Opens Windows Explorer, displaying the scripts directory of the Session.

5.5.3 Media Session node

This section describes the context menu items that are present for a Media Session Node that are not already discussed in 5.5.1 (Session node).

5.5.3.1 Validate Media Files(s)...

Validate a collection of Media file(s) as specified by the user.

Both DICOMDIR file and/or multiple Media files may be validated at once, by selecting them in the presented file browser.

If the checkbox “Generate detailed validation results”, present in the “Session Information” tab, is enabled, both a Summary and Detailed Results will be generated when the validation ends. If the checkbox is disabled, only a Summary Results will be generated.

When a DICOMDIR file has been validated, only the main Summary Results and, if configured for a Session, the main Detailed Results will be visible in the Session Tree. When viewing:

- The main Summary Results, hyperlinks may be present in the “Validation Results” tab, which may be used to navigate to a sub Summary Results. Each sub Summary Results represents the validation results of a DICOMDIR record (and optional referenced file) containing warnings and/or errors. See the figure below for an example.
- The main Detailed Results, hyperlinks may be present in the “Validation Results” tab, which may be used to navigate to a sub Summary Results. Each sub Summary Results represents the validation results of a DICOMDIR record (and optional referenced file).

Session Information Specify SOP Classes Activity Logging Validation Results				
Directory Record TOC				
Record Type	Index	Record Offset	Reference Count	Comments
IMAGE	3	0x000003B8=952	0	Error: Record of related image contains an Error. Number of Err: 4 Number of Wrn: 2
IMAGE	4	0x00000576=1398	0	Error: Record of related image contains an Error. Number of Err: 4 Number of Wrn: 2
IMAGE	5	0x00000734=1844	0	Error: Record of related image contains an Error. Number of Err: 4 Number of Wrn: 2
IMAGE	6	0x000008F2=2290	0	Error: Record of related image contains an Error. Number of Err: 4 Number of Wrn: 2

Example of the main Summary Results for a DICOMDIR with hyperlinks to the sub Summary Results

5.5.3.2 Create DICOMDIR

This option prompts the user for selection of one or more DICOM media files for generation of new DICOMDIR out of selected media files.

5.5.3.3 Create DICOMDIR using Media directory

This option prompts the user for selection of directory/folder contains DICOM media files for generation of new DICOMDIR.

5.5.3.4 Validate DICOMDIR without Reference File

Validate DICOMDIR without validating reference files referred from Image records.

5.5.4 Script node

This section describes the context menu items that are present for a Script node.

5.5.4.1 Edit Script with Notepad...


Edit the selected script with notepad.

See 5.3.2.4 (Menu item “Edit Script with Notepad...”) for more information.

5.5.4.2 Execute

Execute the selected script.

The following sequence of actions will take place in the Project View in which the execution is started:

1. The Session Tree is disabled.
2. If more than one Project View is open, the Session, in which the script is executed, is disabled in the other Project views.
3. The “Activity Logging” tab will become the selected (and only visible) tab. Executing progress may be monitored by inspecting the “Activity Logging” tab.
4. Execution starts.
5. The execution of the script either ends automatically or ends because the user presses the  (“Stop”) button in the toolbar.
6. The “Validation Results” tab becomes the selected tab, in which the contents of the, automatically selected, summary results file is shown.
7. If more than one Project View is open, the Session, in which the script was executed, is enabled again in the other Project views.
8. The Session Tree is enabled.

Instead of using this context menu item, the user can accomplish the same by double-clicking on the Script.

If the checkbox “Generate detailed validation results”, present in the “Session Information” tab, is enabled, both a Summary and Detailed Results will be generated when the execution ends. If the checkbox is disabled, only a Summary Results will be generated.

5.5.4.3 Remove all Results Files

Remove all Results for the selected Script.

5.5.5 Results node

5.5.5.1 Remove

Remove the selected Results.


5.5.6 Emulator node

5.5.6.1 Execute

Execute the selected Emulator.

The following sequence of actions will take place in the Project View in which the execution is started:

1. The Session Tree is disabled.
2. If more than one Project View is open, the Session, in which the Emulator is executed, is disabled in the other Project views.
3. The “Activity Logging” tab will become the selected (and only visible) tab. Executing progress may be monitored by inspecting the “Activity Logging” tab.
4. Execution starts.

5. If the selected Emulator is a SCP emulator, the user has to end execution by pressing the  (“Stop”) button in the toolbar. If it is a SCU emulator, the execution of the emulator ends automatically or ends because the user presses the “Cancel” button.
6. The “Validation Results” tab becomes the selected tab, in which the contents of the, automatically selected, summary Results file is shown.
7. If more than one Project View is open, the Session, in which the Emulator was executed, is enabled again in the other Project views.
8. The Session Tree is enabled.

Instead of using this context menu item, the user can accomplish the same by double-clicking on the Emulator.

If the checkbox “Generate detailed validation results”, present in the “Session Information” tab, is enabled, both a Summary and Detailed Results will be generated when the execution ends. If the checkbox is disabled, only a Summary Results will be generated.

See 5.6 (Emulators) for more information about the actual usage of the different emulators.

5.6 Emulators

The three emulators currently present in the DVT UI are discussed in this section.

An emulator is accessed through an Emulator Session. In the Session tree, use the context menu item “Execute” of one of the Emulator nodes (present under an Emulator Session node) to execute an Emulator, or just double-click on an Emulator node.

Pressing the “Specify TS” button in the “Session Information” tab (button is only present for an Emulator Session), results in a dialog in which the supported Transfer Syntaxes used by the Emulators may be specified.

5.6.1 Storage SCP Emulator

5.6.1.1 General Storage

DVT will act as an SCP for the Storage SOP Classes, which are specified in the “Specify SOP Classes” tab. All communication is displayed in the “Activity Logging” tab.

The Image (Object) Relationship Analysis will be displayed in the “Activity Logging” tab when the emulator is stopped.

The Emulator automatically supports the Verification SOP Class

NOTE: The SOP Classes specified in the “Specify SOP Classes” tab determine which Storage SOP Classes are supported during the Association phase. Any other Storage SOP Class presented by the SCU will automatically be rejected by DVT.

5.6.1.2 Storage Commitment Support

DVT will act as a Storage Commitment SCP emulator if the Storage Commitment PUSH Definition File is loaded into a Storage SCP Emulator Session.

A Storage SCP Emulator Session supporting only Storage Commitment PUSH can be started on a separate TCP/IP port number if necessary.

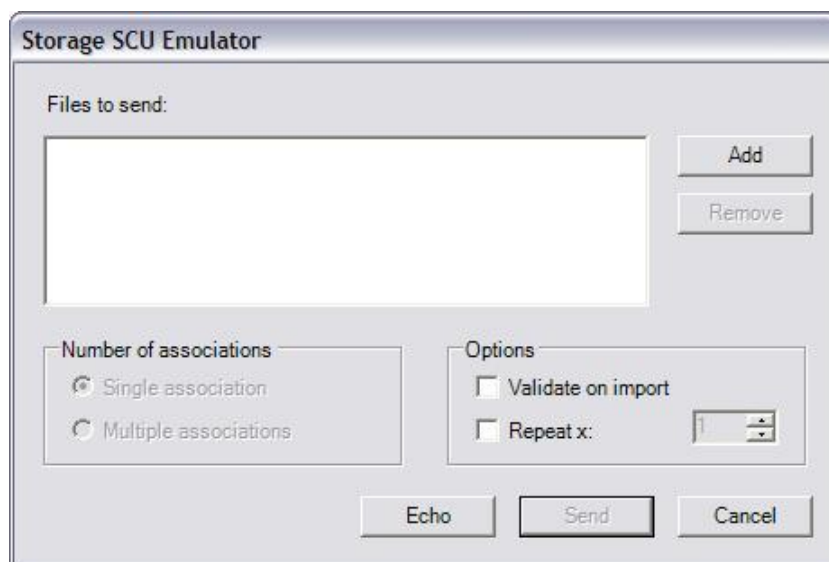
DVT will maintain a list of the SOP Class UID/SOP Instance UID pairs for each storage instance received since the Storage SCP Emulator was started. When the SUT (acting as SCU) issues the N-ACTION-RQ Storage Commitment DVT will check the list of stored instances and flag any being committed for storage. DVT will then respond with a successful N-ACTION-RSP.

DVT will then wait for a fixed delay (currently 5 seconds) before making an association with the SUT in order to send the N-EVENT-REPORT-RQ Storage Commitment. DVT will use the SCP/SCU Role Selection when requesting the association and check that the same is returned by the SUT in the acceptance. DVT will send successful storage commitment for all requested storage instances that are found in the list and failed storage commitment for any request but not found.

NOTE: In order to allow DVT to make the association define the SUT IP address and port number using the SUT ACSE Properties Remote Host Name and Remote Connect Port.

5.6.2 Storage SCU Emulator

DVT will act as a Storage SCU for the Media files selected. The required Storage SOP Classes, for the selected Media files, can be specified in the “Specify SOP Classes” tab.



The “Storage SCU Emulator” dialog

The “Echo” button causes DVT to request a Verification Association and issue a C-ECHO-RQ to the SCP.

Select the Media files that should be sent by the Storage SCU. Use the “Add” and “Remove” buttons to generate the required list. When the “Add” button is pressed, a file browser is presented for selecting extra Media file(s).

The “Send” button is used to start the Storage SCU. The way in which the selected Media files are handled can be controlled by the following parameters.

- **Single/Multiple Association** – Define how the Association will be handled when more than one Media file is selected or the Repeat count is greater than one. DVT will either send all selected Media files during a single Association or request a new Association for each Media file selected.
- **Validate on import** – Validate the Media file against the appropriate Definition File during import (before sending the Media file to SCP).
- **Repeat x 1** – Indicate the number of times that the selected Media files should be sent to the SCP. This is useful when running a duration test on the SCP.

NOTE: DVT will automatically modify the SOP Instance UID (0008,0018) for the Media files selected when the repeat count is greater than one. This allows the same Media file content to be sent as many times, as indicated by the count, but to be seen by the SCP as a different instance each time.

5.6.3 Print SCP Emulator

DVT will act as an SCP for a Print SOP class (the actual definition file(s) used are specified in the “Specify SOP Classes” tab). It is recommended to work with one Print Meta SOP Class at a time.

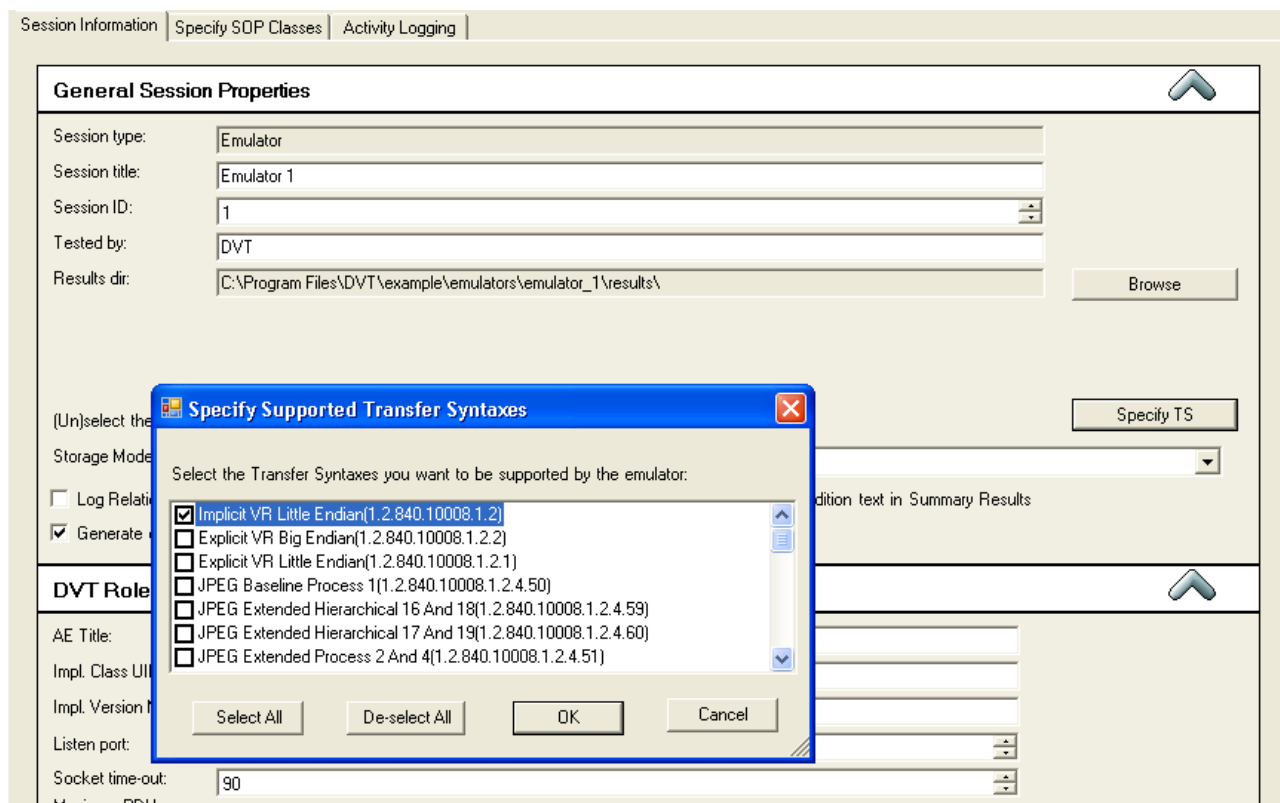
If the Print Job SOP Class Definition File is loaded, DVT will support the Print Job Service Class.

If “SLIDE”, “SUPERSLIDE” or “CUSTOM” Image Display Formats [DICOM tag (2010,0010)] are to be sent, the number of images for each format must be defined in the “ImageDisplayFormat.dat” data. This file must also be modified to indicate the number of annotation boxes defined if the Annotation Display Format ID [DICOM tag (2010,0030)] is to be used.

See 5.3.4.1 (Menu item “Status Print SCP Emulator”) for more information on how to set the Printer Status.

5.6.4 Emulator Transfer Syntaxes

The Emulator Transfer Syntaxes dialog box is displayed as follows. The User should select the Transfer Syntaxes that are to be supported by the emulator.



The button Specify TS (**SUPPORTED-TRANSFER-SYNTAX** <uid> parameter) is used to define the Transfer Syntaxes that DVT will accept when working in emulation mode. This parameter is repeated for each Transfer Syntax supported by DVT.

When an SCU emulator is being used, DVT will propose all the chosen Transfer Syntaxes once per SOP Class being used – i.e., If Implicit VR Little Endian and Explicit VR Big Endian are chosen, then DVT will propose two Presentation Contexts for each SOP Class being used. DVT does not combine the Transfer Syntaxes into a single Presentation Context, but allows the SCP to decide which will be supported during the Association.

When an SCP emulator is being used, DVT will accept Presentation Contexts from the SCU where the proposed SOP Class/Transfer Syntax matches the Transfer Syntaxes chosen for the SOP Classes being used.

When the User starts a New Emulator Session, the Implicit VR Little Endian Transfer Syntax is selected as the default value.

6. Command Line

The Command-line version of DVT has the following usage:

Usage:

DvtCmd [-v]

<Full session file name>

<Script file name (session scriptdirectory used when file name only)>

Script filename must end with extension .dss or .vbs

or

DvtCmd -c

<Full script file name>

Script file name must end with extension .vbs

or

DvtCmd -m

<Full session file name>

<Full media file name>

or

DvtCmd -estscp

<Full session file name>

or

DvtCmd -estscu

<Full session file name>

<List of full media file names>

<Multiple/Single association option>

<Validate on Import option>

<Send data under new study option>

<Nr. of Repetitions>

Media files can also contain DICOMDIR.

The third, fourth, fifth and sixth parameters are optional.

The value of the third, fourth and fifth parameters should be provided true/false

or

DvtCmd -eprscp

<Full session file name>

<Printer status>

<Printer status info>

Options:

-c: Only Compile, No Execute

-m: Validate media file

-estscp: Run SCP Storage emulator
-estscu: Execute SCU Storage emulator
-eprscp: Run SCP Printer emulator
-v: Verbose
-h: Help

The VBS scripts use entry-point Sub: DvtCmdMain in Module: DvtnScript

Results of running the Command-line will be written to Results Files as with the GUI version.

7. Programming DVT

7.1 General

DVT interprets the contents of the DICOMScript files in order to perform a validation. DVT is “programmed” by writing DICOMScripts that correspond with the given Test Scenario. Simple DICOMScripts can be written using the following two commands:

- SEND
- RECEIVE

See section on Advanced Programming for details of the other commands supported.

DVT can SEND and RECEIVE DICOM Messages according to the sequence defined in the DICOMScript. The following messages can be handled:

- ACSE Requests & Responses.
- DIMSE Messages.

When sending messages, the parameters / attributes defined by the given DICOMScript are encoded. In addition, for DIMSE Messages, the corresponding Definition can be consulted to “complete” the Message (encode any Type 2 attributes with zero length).

Any parameter / attribute value can be set for sending.

The ability to set any parameter / attribute value is extremely useful when testing the robustness of a SUT’s implementation.

Error and Exception Handling can easily be controlled - the SUT’s reaction to such events gives an indication of the quality of the implementation.

When receiving messages, the parameters / attributes received from the SUT are first checked against the corresponding Definition and then against those defined by the given DICOMScript.

NOTE: *If the received message is different from that defined in the DICOMScript, the validation will fail even if the received message conforms to DICOM - this is because the Test Scenario has failed.*

NOTE: *Since DVT uses the backslash character (“\”) as a prefix for a HEX value (see section 7.7, “Extended Character Sets”), all backslashes in the script must be entered as a double backslash (“\\”) to be correctly read by DVT. The exceptions to this are filenames and for the Image Display Format, DICOM tag (2010, 0010), which will allow use of a single backslash.*

7.2 Send, Receive ACSE Requests and Responses

DVT supports both sending and receiving of all 6 ACSE requests / responses:

- Associate Request
- Associate Accept
- Associate Reject
- Release Request
- Release Response
- Abort Request

Data Transfer is handled at the DIMSE / IOD level - See section Send, Receive DICOM Messages.

7.2.1 Associate Request

The Associate Request has the following syntax:

SEND / RECEIVE ASSOCIATE-RQ <assoc_rq_param_list>

```
<assoc_rq_param_list> (
    PROTOCOL-VERSION <integer>
    CALLED-AE-TITLE <string>
    CALLING-AE-TITLE <string>
    APPLICATION-CONTEXT-NAME <string>
    PRESENTATION-CONTEXT-ITEMS
        (<sop class uid/name1>, <transfer syntax>)
        (<sop class uid/name2>, <transfer syntax 1>, .., <transfer syntax n>)
    ...
    MAXIMUM-LENGTH-RECEIVED <integer>
    IMPLEMENTATION-CLASS-UID <string>
    IMPLEMENTATION-VERSION-NAME <string>
    SOP-CLASS-EXTENDED-NEGOTIATION
        (<sop class uid/name1>, <integer>(applInfo1), .., <integer>(applInfon))
        (<sop class uid/name2>, <integer>(applInfo1), .., <integer>(applInfon))
    ...
    SCPSCU-ROLE-SELECTION
        (<sop class uid/name1>, <integer>(scpRole), <integer>(scuRole))
        (<sop class uid/name2>, <integer>(scpRole), <integer>(scuRole))
    ...
    ASYNCHRONOUS-OPERATIONS-WINDOW <integer>(invoked) <integer>(performed)
)
```

The standard Associate Request parameters can take default values except for the **PRESENTATION-CONTEXT-ITEMS**, which must be provided explicitly.

The **SOP-CLASS-EXTENDED-NEGOTIATION**, **SCPSCU-ROLE-SELECTION** and **ASYNCHRONOUS-OPERATIONS-WINDOW** are optional and must be provided by the User in the DICOMScript.

The following parameters have “in-built” default values as shown:

PROTOCOL-VERSION	1
CALLED-AE-TITLE	“DVT_AE”
CALLING-AE-TITLE	“DVT_AE”
APPLICATION-CONTEXT-NAME	“1.2.840.10008.3.1.1.1”
MAXIMUM-LENGTH-RECEIVED	16384
IMPLEMENTATION-CLASS-UID	“1.2.826.0.1.3680043.2.1545.1”
IMPLEMENTATION-VERSION-NAME	“dvtVersionMajor.VersionMinor”

NOTE: The **IMPLEMENTATION-CLASS-UID** and **IMPLEMENTATION-VERSION-NAME** include of the DVT software version number. This consists of a major version number and minor version number.

The **CALLED-AE-TITLE**, **CALLING-AE-TITLE**, **MAXIMUM-LENGTH-RECEIVED**, **IMPLEMENTATION-CLASS-UID** and **IMPLEMENTATION-VERSION-NAME** parameters can be further defined in the Session File.

These parameter values are checked in the following order. The first value found will be used.

- 1) DICOMScript specific value
- 2) Session File general value
- 3) In-built default value

The **PRESENTATION-CONTEXT-ITEMS** are the Proposed Abstract/Transfer Syntax combinations for the Association:

```
(<sop class uid/name>, <transfer syntax 1>, .., <transfer syntax n>)
```

<sop class uid/name> is the Abstract Syntax Name. Value is either a UID or the text equivalent from the corresponding Definition File - see (META)SOPCLASS.

<transfer syntax> is the Transfer Syntax Name. Value is either any valid Transfer Syntax UID or the following frequently-used text equivalent:

“Implicit VR Little Endian”
 “Explicit VR Little Endian”
 “Explicit VR Big Endian”

Example 1:

```
SEND ASSOCIATE-RQ (
  PRESENTATION-CONTEXT-ITEMS
  (“XA Image Storage SOP Class”,
    “Explicit VR Big Endian”,
    “Explicit VR Little Endian”,
    “Implicit VR Little Endian”)
  (“Basic Grayscale Print Management Meta SOP Class”,
    “1.2.840.10008.1.2”)
)
```

DVT proposes XA Image Storage SOP Class and Basic Grayscale Print Management Meta SOP Class - XA with 3 possible transfer syntaxes, Print with only Implicit VR Little Endian (UID). Take all other parameters via default mechanism.

Example 2:

```
SEND ASSOCIATE-RQ (
  PROTOCOL-VERSION 2
  PRESENTATION-CONTEXT-ITEMS
  (“CT Image Storage SOP Class”, “Implicit VR Little Endian”)
)
```

DVT proposes CT Image Storage SOP Class with only Implicit VR Little Endian. Take all other parameters via default mechanism, except for PROTOCOL-VERSION which is explicitly set to 2.

7.2.2 Associate Accept

The Associate Accept has the following syntax:

SEND / RECEIVE ASSOCIATE-AC <assoc_ac_param_list>

```
<assoc_ac_param_list> (
  PROTOCOL-VERSION <integer>
  CALLED-AE-TITLE <string>
  CALLING-AE-TITLE <string>
  APPLICATION-CONTEXT-NAME <string>
  PRESENTATION-CONTEXT-ITEMS
    (<sop class uid/name1>, <result/reason>, <accepted transfer syntax>)
    (<sop class uid/name2>, <result/reason>)
    ...
  MAXIMUM-LENGTH-RECEIVED <integer>
  IMPLEMENTATION-CLASS-UID <string>
  IMPLEMENTATION-VERSION-NAME <string>
  SOP-CLASS-EXTENDED-NEGOTIATION
    (<sop class uid/name1>, <integer>(applInfo1),...,<integer>(applInfoN))
    (<sop class uid/name2>, <integer>(applInfo1),...,<integer>(applInfoN))
    ...
  SCPSCU-ROLE-SELECTION
    (<sop class uid/name1>, <integer>(scpRole),<integer>(scuRole))
    (<sop class uid/name2>, <integer>(scpRole),<integer>(scuRole))
    ...
  ASYNCHRONOUS-OPERATIONS-WINDOW <integer>(invoked) <integer>(performed)
)
```

The standard Associate Accept parameters can take default values except for the **PRESENTATION-CONTEXT-ITEMS**, which must be provided explicitly.

The **SOP-CLASS-EXTENDED-NEGOTIATION**, **SCPSCU-ROLE-SELECTION** and **ASYNCHRONOUS-OPERATIONS-WINDOW** are optional and must be provided by the User in the DICOMScript.

The following parameters have “in-built” default values as shown:

PROTOCOL-VERSION	1
CALLED-AE-TITLE	“DVT_AE”
CALLING-AE-TITLE	“DVT_AE”
APPLICATION-CONTEXT-NAME	“1.2.840.10008.3.1.1.1”
MAXIMUM-LENGTH-RECEIVED	16384
IMPLEMENTATION-CLASS-UID	“100.118.116.Year.VersionMajor.VersionMinor”
IMPLEMENTATION-VERSION-NAME	“dvtVersionMajor.VersionMinor”

The **CALLED-AE-TITLE**, **CALLING-AE-TITLE**, **MAXIMUM-LENGTH-RECEIVED**, **IMPLEMENTATION-CLASS-UID** and **IMPLEMENTATION-VERSION-NAME** parameters can be further defined in the Session File.

These parameter values are checked in the following order. The first value found will be used.

- 4) DICOMScript specific value
- 5) Session File general value
- 6) In-built default value

The **PRESENTATION-CONTEXT-ITEMS** are the Accepted Abstract/Transfer Syntax combinations for the Association:

(*<sop class uid/name>*, *<result/reason>*, *<accepted transfer syntax >*)

<sop class uid/name> is the Abstract Syntax Name. Value is either a UID or the text equivalent from the corresponding Definition File - see (META)SOPCLASS.

<result/reason> is the Result/Reason field:

- 0** = *acceptance*
- 1** = *user rejection*
- 2** = *no reason (provider rejection)*
- 3** = *abstract syntax not supported (provider rejection)*
- 4** = *transfer syntaxes not supported (provider rejection)*

<accepted transfer syntax> is the Transfer Syntax Name. Value is either a valid Transfer Syntax UID or the following frequently-used text equivalent:

- “Implicit VR Little Endian”**
- “Explicit VR Little Endian”**
- “Explicit VR Big Endian”**

NOTE: The Transfer Syntax Name should only be present when the Result/Reason field equals “acceptance”. The value of the Transfer Syntax Name identifies the transfer syntax that the SCP wants to use for the given Abstract Syntax Name (SOP Class UID).

Example: Possible response to Associate Request Example 1:

```
SEND ASSOCIATE-AC (
  PRESENTATION-CONTEXT-ITEMS
  (“XA Image Storage SOP Class”, 0, “Implicit VR Little Endian”)
  (“Basic Grayscale Print Management Meta SOP Class”, 3)
)
```

Accept the XA Image Storage SOP Class with Implicit VR Little Endian transfer syntax. Reject the Basic Grayscale Print Management Meta SOP Class with Result/Reason “abstract syntax not supported”.

7.2.3 Associate Reject

The Associate Reject has the following syntax:

SEND / RECEIVE ASSOCIATE-RJ <assoc_rj_param_list>

```
<assoc_rj_param_list> (
    RESULT <integer>
    SOURCE <integer>
    REASON <integer>
)
```

RESULT:

- 1** = *rejected permanent*
- 2** = *rejected transient*

SOURCE:

- 1** = *DICOM UL service user*
- 2** = *DICOM UL service provider (ACSE related function)*
- 3** = *DICOM UL service provider (Presentation related function)*

REASON: (in combination with SOURCE)

SOURCE = **1**:

- 1** = *no reason given*
- 2** = *application context name not supported*
- 3** = *calling AE title not recognized*
- 7** = *called AE title not recognized*

SOURCE = **2**:

- 1** = *no reason given*
- 2** = *protocol version not supported*

SOURCE = **3**:

- 1** = *temporary congestion*
- 2** = *local limit exceeded*

All Associate Reject parameters have “in-built” default values as shown:

RESULT	1 (rejected permanent)
SOURCE	2 (DICOM UL service provider (ACSE related function))
REASON	1 (no reason given)

Example: Possible response to Associate Request Example 2:

```
SEND ASSOCIATE-RJ (
    RESULT 1
    SOURCE 2
    REASON 2
)
```

Reject Associate Request with “protocol version not supported”.

7.2.4 Release Request

The Release Request has the following syntax:

SEND / RECEIVE RELEASE-RQ

Release Request does not have any parameters.

7.2.5 Release Response

The Release Response has the following syntax:

SEND / RECEIVE RELEASE-RP

Release Response does not have any parameters.

7.2.6 Abort Request

The Abort Request has the following syntax:

SEND / RECEIVE ABORT-RQ <abort_rq_param_list>

```
<abort_rq_param_list> (
    SOURCE <integer>
    REASON <integer>
)
```

SOURCE:

0 = DICOM UL service user (initiated abort)
2 = DICOM UL service provider (initiated abort)

REASON:

0 = reason not specified	4 = unrecognized PDU parameter
1 = unrecognized PDU	5 = unexpected PDU parameter
2 = unexpected PDU	6 = invalid PDU parameter value

The Abort parameters have “in-built” default values as shown:

SOURCE	2 (DICOM UL service provider (initiated abort))
REASON	0 (reason not specified)

Example:

```
SEND ABORT-RQ (
    SOURCE 2
    REASON 6
)
```

SCP aborts association due to “invalid pdu parameter value”.

7.3 Send, Receive DICOM Messages

7.3.1 General

DVT supports both sending and receiving of all DIMSE Commands - Requests and Responses.

IOD Support is provided by loading the appropriate Definition File. The DIMSE Command / IOD combination must match one of the DEFINES in the Definition File.

The following tokens are used in the syntax description.

<i><integer></i>	Integer value applicable to the parameter being assigned.
<i><hex></i>	Integer value expressed in hexadecimal (e.g., 0x01AF).
<i><string></i>	Character String enclosed in double quotes (“”).
<i><string n></i>	Character String enclosed in double quotes (“”) with a maximum length of ‘n’ characters.
<i><uid></i>	DICOM UID enclosed in double quotes (“”).
<i><command></i>	DICOM Command: C-ECHO-RQ, C-ECHO-RSP, C-FIND-RQ, C-FIND-RSP, C-GET-RQ, C-GET-RSP, C-MOVE-RQ, C-MOVE-RSP, C-STORE-RQ, C-STORE-RSP, C-CANCEL-RQ, N-ACTION-RQ, N-ACTION-RSP, N-CREATE-RQ, N-CREATE-RSP, N-DELETE-RQ, N-DELETE-RSP, N-GET-RQ, N-GET-RSP, N-SET-RQ, N-SET-RSP, N-EVENT-REPORT-RQ, N-EVENT-REPORT-RSP
<i><dataset_name></i>	Character String enclosed in double quotes (“”) that matches the name of a Dataset in the Definition File.
<i><item_name></i>	Character String enclosed in double quotes (“”) that matches the name of an Item in the Definition File.
<i>< identifier></i>	Identifier String, not enclosed in double quotes (“”). An identifier is used to identify an instance of the given object. Maximum length 64 characters. An identifier may contain any character in the range A-Z, 0-9 and underscores ‘_’
<i><command_ref></i>	Reference to a Command DICOMObject: <i><command> [<identifier>]</i>
<i><dataset_ref></i>	Reference to a Dataset DICOMObject: <i><dataset_name> <identifier></i>
<i><item_ref></i>	Reference to an Item DICOMObject: <i><item_name> <identifier></i>
<i><dicom_object_ref></i>	Reference to a DICOMObject: <i><command_ref> / <dataset_ref> / <item_ref></i>
<i><tag></i>	DICOM Attribute Tag - in hexadecimal (e.g., 0x00100010).

<vr>	DICOM Attribute VR: AE, AS, AT, CS, DA, DS, DT, FD, FL, IS, LO, LT, OB, OF, OW, PN, SH, SL, SQ, SS, ST, TM, UI, UL, UN, US, UT.
<value>	DICOM Attribute Value: <string> - for “String” based VR types; <hex> for hex based VR Types; <integer> for integer based VR types; <identifier> for SQ based VR type (identifier of item to be included in sequence); <uid> for UI based VR type.

NOTE: The Value representations *US\US*, *SS\US* are not supported. Most of these variable VR types are handled by DVT internally.

7.3.2 Send, Receive

Definition File for XA Image Storage SOP Class:

DEFINE C-STORE-RQ “XA Image”

Possible use in DICOMScript:

SEND C-STORE-RQ “XA Image”

Command Syntax:

SEND / RECEIVE <command> [<dicom_object_attribute_list>]

<dicom_object_attribute_list> (
<attribute 1>
..
<attribute n>
)

<attribute> Attribute of the Command.

Command and Dataset Syntax:

SEND / RECEIVE <command> <dataset_name> [<dicom_object_attribute_list>]

<dicom_object_attribute_list> (
<attribute 1>
..
<attribute n>
)

<attribute> Attribute of Command or Dataset.

Attribute syntax - <attribute>:

(<tag>, <vr>, <value 1>, .., <value n>)
 or
(<tag>, <value 1>, .., <value n>)

NOTE: The VR can be enclosed in brackets with the following meaning:

(VR) – force VR to type ‘UN’ during DICOMObject encoding.

(VR?) – force VR to type ‘??’ during DICOMObject encoding. The type ‘??’ is used by some DICOM libraries as a representation of the Unknown VR ‘UN’.

<value> Attribute Value appropriate for VR.

When the <vr> is not defined in the DICOMScript, the <tag> must be known in the Definition File - enables <vr> look-up.

Table 9.1 : Value Representations					
VR	Name	Max. Length	VR	Name	Max. Length
AE	AE Title	16 bytes	OW	Other Word String	-
AS	Age String	4 bytes	PN	Person Name	64 chars per component group
AT	Attribute Tag	4 bytes	SH	Short String	16 chars
CS	Code String	16 bytes	SL	Signed Long	4 bytes
DA	Date	8 bytes	SQ	Sequence of Items	-
DS	Decimal String	16 bytes	SS	Signed Short	2 bytes
DT	Date / Time	26 bytes	ST	Short Text	1024 chars
FL	Floating Point Single	4 bytes	TM	Time	16 bytes
FD	Floating Point Double	8 bytes	UI	Unique Identifier	64 bytes
IS	Integer String	12 bytes	UL	Unsigned Long	4 bytes
LO	Long String	64 chars	UN	Unknown	-
LT	Long Text	10240 chars	US	Unsigned Short	2 bytes
OB	Other Byte String	-	UT	Unlimited Text	2**32-2
OF	Other Float String	-			

The attribute values must be used according to the “base” type of the value - String, Integer and Hexadecimal.

String based values.

The following Attribute VR values should be used as strings:

(<tag>, AE, <string>)	“AE Title”
(<tag>, AS, <string>)	“020Y”
(<tag>, CS, <string>)	“PRIMARY”
(<tag>, DA, <string>)	“19970109”
(<tag>, DS, <string>)	“+1.0359” or “-1.03e-4”
(<tag>, DT, <string>)	“970109101735.002”
(<tag>, FL, <string>)	“+1.0359”
(<tag>, FD, <string>)	“-20.34”
(<tag>, IS, <string>)	“123” or “-123” or “+123”
(<tag>, LO, <string>)	
(<tag>, LT, <string>)	
(<tag>, PN, <string>)	
(<tag>, SH, <string>)	
(<tag>, ST, <string>)	
(<tag>, TM, <string>)	“101735”
(<tag>, UI, <string>)	“1.2.840.10008.1.1” or text equivalent (look-up)
(<tag>, UN, <string>)	
(<tag>, UT, <string>)	
(<tag>, OB, <string>)	“ob_data.pix”
(<tag>, OF, <string>)	“of_data.pix”
(<tag>, OW, <string>)	“ow_data.pix”

The PN VR should be encoded as defined in [1] part 5. DVT supports the 3 component group format of the PN. This should be encoded as a single string with the equals character “=” (3DH) between the component groups. The User should also apply the caret “^” (5EH) component delimiter where appropriate. DVT will validate the format of any PN attributes.

Other VR values expressed as String will result in an error message from DVT during DICOMScript parsing.

Note: The maximum number of bytes that can be entered for a string value is 1024.

Integer based values.

The following Attribute VR values should be used as integers:

(<tag>, SS, <integer>)	10359 or -10359
(<tag>, SL, <integer>)	103590 or -103590
(<tag>, US, <integer>)	10359
(<tag>, UL, <integer>)	103590
(<tag>, OB, <integer>)	512 - defines pattern 512 X 512 X 8bit
(<tag>, OF, <integer>)	128 - defines pattern 128 X 128 X 32bit (floating point)
(<tag>, OW, <integer>)	1024 - defines pattern 1024 X 1024 X 16bit

Other VR values expressed as Integer will result in an error message from DVT during DICOMScript parsing.

Hexadecimal based values.

The following Attribute VR values should be used as hexadecimal:

(<tag>, AT, <hex>)	0x00100010
(<tag>, US, <hex>)	0x01FF
(<tag>, UL, <hex>)	0x02ABC10
(<tag>, OB, <hex>)	0x200 - defines pattern 512 X 512 X 8bit
(<tag>, OF, <hex>)	0x80 - defines pattern 128 X 128 X 32bit (floating point)
(<tag>, OW, <hex>)	0x400 - defines pattern 1024 X 1024 X 16bit

Other VR values expressed as Hexadecimal will result in an error message from DVT during DICOMScript parsing.

Example 1:

```
SEND C-STORE-RQ "CT Image" (
...
(0x00100010, "Doe^John")
...
(0x00080008, "ORIGINAL", "PRIMARY", "LOCALIZER")
...
(0x00280004, "MONOCHROME1")
(0x00280010, 0x200)
(0x00280011, 0x200)
...
(0x7FE00010, OB, "ct-img.dat")
...
)
```

Send a CT Image for patient John Doe, taking the pixel data from file "ct-img.dat".

NOTE: The CT Image example is incomplete.

Example 2:

```
RECEIVE C-STORE-RSP (
(0x00000900, 0x0000)
)
```

Receive a C-STORE-RSP with STATUS 0x0000 (OK) indicating that store operation was successful.

7.3.3 Sequence Syntax

When using Sequences within the DICOMScript, each item is encoded as a separate <attribute>, nested within the <value> of the Sequence Attribute by using the '>' character. This syntax allows items to be defined "by value" within the parent sequence (*DVT also supports item definition "by reference" – see section on Advanced Programming.*)

A zero-length Sequence is encoded as follows:

```
(<tag>, SQ, "")
```

The Sequence will have a zero-length – there will be no items encoded.

A Sequence with a single zero-length item is encoded as follows:

```
(<tag>, SQ, )
```

A Sequence with 3 zero-length items is encoded as follows:

```
(<tag>, SQ, , , )
```

A Sequence may also reference items previously created in the Data Warehouse (see section on Advanced Programming for more details):

```
(<tag>, SQ, "ItemId1","ItemId2", "ItemId3")
```

Example 1:

SEND N-CREATE-RSP "Basic Film Box" (

```
...
(0x20100510, SQ,
>(0x00081150, UI, "Basic Grayscale Image Box SOP Class")
>(0x00081155, UI, "1.2.3.4.1")
,
>(0x00081150, UI, "Basic Grayscale Image Box SOP Class")
>(0x00081155, UI, "1.2.3.4.2")
)
...
)
```

Return two items in the Referenced Image Box Sequence, one item for each Image Box in the Film Box.

Example 2:

SEND N-CREATE-RSP "Basic Film Box" (

```
...
(0x20100510, SQ, "")
...
)
```

Send a zero-length Referenced Image Box Sequence.

7.4 Pattern Generation

DVT can generate various data patterns for the following VR types: OB/OF/OW and ST/LT/UT.

7.4.1 OB/OF/OW Pattern Generation

OB/OF/OW patterns can be generated during DICOMScript execution. The syntax to define OB/OF/OW data patterns is:

(<tag>, OB|OF|OW, **rows**, **columns**, **startValue**, **rowIncrement**, **columnIncrement**, **rowsSame**, **columnsSame**)

where:

- **rows** – # rows in pattern.
- **columns** – # columns in pattern.
- **startValue** – top/left pattern value expressed in pixels.
- **rowIncrement** – **startValue** is incremented by the **rowIncrement** to produce successive row pattern values.
- **columnIncrement** – **startValue** is incremented by the **columnIncrement** to produce successive column pattern values.
- **rowsSame** – # row values which are the same before the **rowIncrement** is added.
- **columnsSame** – # column values which are the same before the **columnIncrement** is added.

The following examples will illustrate some patterns that can be produced for a 512 X 512 OW pattern (i.e., rows = 512 and columns = 512)

Example 1:

(0x7FE00010, OW, 512, 512, 0, 8, 0, 1, 1)

Result: OW data – shading from top to bottom, min pixel to max pixel, step 8.

Example 2:

(0x7FE00010, OW, 512, 512, 0, 0, 8, 1, 1)

Result: OW data – shading from left to right, min pixel to max pixel, step 8.

Example 3:

(0x7FE00010, OW, 512, 512, 0, 65535, 0, 256, 1)

Result: OW data – 2 horizontal stripes, 1st min pixel, 2nd max pixel.

Example 4:

(0x7FE00010, OW, 512, 512, 65535, 65535, 65535, 64, 64)

Result: OW data – 8 X 8 checker board, min pixel, max pixel.

Example 5:

(0x7FE00010, OW, 512, 512, 2048, 0, 0, 1, 1)

Result: OW data – flat field, half max pixel.

7.4.2 ST/LT/UT Pattern Generation

ST/LT/UT patterns can be generated during DICOMScript execution. The syntax to define ST/LT/UT data patterns is:

(<tag>, ST|LT|UT, **length**)

where:

- **length** – length of data pattern to generate. The pattern contains sequences of letters a..z terminated with a newline.

The following examples will illustrate some patterns that can be produced.

Example 1:

(0x00082111, ST, 60)

Result: DerivationDescription of “abcdefghijklmnopqrstuvwxyz\nabcdefghijklmnopqrstuvwxyz\nabcdef”.

Example 2:

(0x00104000, LT, 10)

Result: Patient History of “abcdefghij”.

7.5 Using Labels - Value Mapping

In some cases the SUT will generate Attribute values which have to be used later in the DICOMScript to continue the scenario. The User labels an Attribute value generated by the SUT (at the position in the DICOMScript) where it is received. DVT will maintain a mapping between the label and the actual Attribute value received from the SUT. DVT will then translate any further occurrences of the label into the corresponding Attribute value.

The keyword **LABEL:** is used in the DICOMScript to indicate the position at which the mapping is to start. The following syntax is allowed:

- “**LABEL:Name**” – Define “**Name**” as label for corresponding Attribute.
- “**LABEL:Name:Value**” – Define “**Name**” as label for corresponding Attribute and assign it “**Value**”.

The “Value” can also be a previously used “Name” so that the Attribute value associated with the “Name” gets copied to the “Value” - example:

(0x00321032, “**LABEL:ReqPhysName:Requesting^Dr**”)

...

(0x00080090, “**LABEL:RefPhysName:ReqPhysName**”)

- can be used to set the initial value of (0032,1032) to “**Requesting^Dr**”, later the same value being copied to (0008,0090). (This is useful when testing the Modality Worklist and Performed Procedure Step SOP Classes.)

The keyword **NEW:** is used in the DICOMScript to indicate that DVT should generate a new attribute value and map it to the given label. DVT will translate any further occurrences of the label into the new attribute value.

The **NEW:** and **LABEL:** keywords can be used with VR types **LO**, **SH** & **UI**. The **LABEL:** keyword can be used with VR types **AE**, **CS**, **DA**, **PN** & **TM**.

***NOTE:** The keywords **LABEL:** and **NEW:** form part of the actual labeled string.*

This feature is found in DICOM Print Management where the SUT (Printer) is responsible for generating the UIDs for each Image Box belonging to a given Film Box.

Example 1:

SEND N-CREATE-RSP “Basic Film Box” (

...

```
(0x20100510, SQ,
>(0x00081150, UI, "Basic Grayscale Image Box SOP Class")
>(0x00081155, UI, "NEW:ImageBoxId1")
)
...
)
```

RECEIVE N-SET-RQ "Basic Grayscale Image Box" (

```
...
(0x00000003, UI, "Basic Grayscale Image Box SOP Class")
(0x00001001, UI, "ImageBoxId1")
...
```

DVT sets up a mapping between "*NEW:ImageBoxId1*" and the a new UID generated. DVT will then translate any further occurrences of "*ImageBoxId1*" to the new UID.

The use of the NEW: allows the same DICOMScript to be repeated many times in a test scenario by ensuring that a new UID is generated each time.

Example 2:

RECEIVE N-CREATE-RSP "Basic Film Box" (

```
...
(0x20100510, SQ,
>(0x00081150, UI, "Basic Grayscale Image Box SOP Class")
>(0x00081155, UI, "LABEL:ImageBoxId2")
)
...
)
```

SEND N-SET-RQ "Basic Grayscale Image Box" (

```
...
(0x00000003, UI, "Basic Grayscale Image Box SOP Class")
(0x00001001, UI, "ImageBoxId2")
...
)
```

DVT sets up a mapping between "*LABEL:ImageBoxId2*" and the actual UID received from the SUT. DVT will then translate any further occurrences of "*ImageBoxId2*" to the mapped UID.

The use of the LABEL: allows the same DICOMScript to be repeated many times in a test scenario by ensuring that the UID mapping is (re)started at the correct place in the DICOMScript.

7.6 VR Keywords

DVT uses certain keywords in combination with the VR as variables.

The keywords **YESTERDAY**, **TODAY** & **TOMORROW** when encountered by DVT as DA / DT attributes values are translated into the corresponding calendar date (as available on the PC).

The keyword **NOW** when encountered by DVT as TM / DT attributes is translated into the current time (as available on the PC).

The keywords **CALLINGAETITLE** and **CALLEDAETITLE** when encountered by DVT as AE attribute values are translated into the CALLING-AE-TITLE and CALLED-AE-TITLE SUT ACSE Property values.

The keyword **AUTOSET** (not enclosed in double quotes) can be used with attributes of VR DA, TM, DT and UI to set the attribute value to the current date, time, date/time and new UID respectively.

Examples:

(0x00080020, DA, "TODAY")	- use today's date.
(0x00080020, DA, "YESTERDAY-TOMORROW")	- make date range.
(0x00080020, DA, "TODAY-")	- date range from today.

(0x00080030, TM, "NOW")	- use current time.
(0x00080054, AE, "CALLINGAETITLE")	- use ACSE Property CALLING-AE-TITLE.
(0x00080020, DA, AUTOSET)	- use today's date.
(0x0020000D, UI, AUTOSET)	- generate a new UID for the Study Instance.

These VR Keywords are particularly useful when performing Query, Worklist and Detached Management Service Tests.

7.7 Extended Character Sets

DVT checks all appropriate "string" based VR types for extended characters - LO, LT, PN, SH & ST.

When displaying an attribute value containing an extended character, DVT does NOT attempt to use the correct character set. It does, however, display the HEX values of any extended characters after the attribute value. The ESCape sequences are translated by DVT to identify the Character Sets being used.

If extended characters are detected in any of the received attribute values, DVT goes on to check for the presence of the **Specific Character Set** attribute (0008,0005) and that the ESCape sequences encountered reference character sets defined by the Specific Character Set attribute value(s).

If present a WARNING is displayed (asking the user to confirm the value if the Specific Character Set attribute). If not present an ERROR is displayed.

NOTE: The default Extended Character Set is "ISO_IR 100".

Extended characters can be used in the DICOMScripts by prefixing the 2 digit HEX value of the extended character by a **BACKSLASH** (5CH) at the appropriate location in the string.

e.g. "B\D8^John" would be interpreted by DVT as hex string 42D85E4A6F686E.

7.8 Abstract Storage Service

As general Storage Test Scenarios are all the same, varying only in the name of the Storage Service Class, DVT makes use of an Abstract Storage Class. This allows the User to generate a single set of Storage SCP DICOMScripts using the Abstract Storage Service and apply them to all Storage SOP Classes.

The first Storage Definition file loaded in a given Test Session becomes the Abstract Storage Class. DICOMScripts will reference the concrete class via the abstract names.

In ASSOCIATE-RQ / AC:

PRESENTATION-CONTEXT-ITEMS
("Abstract Storage SOP Class", ...)

In DICOMScripts:

SEND / RECEIVE C-STORE-RQ "Abstract Storage IOD"

If the "ct-img.def" Definition is the first loaded, then the abstract names will be translated to the concrete "CT" names by DVT:

"Abstract Storage SOP Class"	->	"CT Image Store SOP Class"
"Abstract Storage IOD"	->	"CT Image"

7.9 Abstract Print Service

As the general Basic Print Test Scenarios are all the same, varying only in the name of the (Meta) Service Class, DVT makes use of an Abstract Print Class.

The first Basic Print Definition file loaded in a given Test Session becomes the Abstract Print Class. DICOMScripts will reference the concrete class via the abstract names.

In ASSOCIATE-RQ / AC:

PRESENTATION-CONTEXT-ITEMS
 ("Abstract Meta SOP Class", ...)

In DICOMScripts:

```
SEND / RECEIVE N-CREATE-RSP "Basic Film Box" (
(0x20100510, SQ,
>(0x00081150, UI, "Abstract Image Box SOP Class")
>(0x00081155, UI, "1.3.51.1.1.1")
)
)
```

```
SEND / RECEIVE N-SET-RQ "Abstract Image Box"
```

If the "gry-prnt.def" Definition is the first loaded, then the abstract names will be translated to the concrete "Basic Grayscale" names by DVT:

"Abstract Meta SOP Class"	->	"Basic Grayscale Print Management Meta SOP Class"
"Abstract Image Box SOP Class"->		"Basic Grayscale Image Box SOP Class"
"Abstract Image Box"	->	"Basic Grayscale Image Box"

7.10 Private Attributes

DVT can be programmed to handle Private Attributes in two ways:

- Modified Definition Files - A Private Definition File can be used to specify the private attribute in the same way as any standard attribute. When the private attribute is used in a DICOMScript, DVT will consult the Definition File for the VR, VM, allowed values and attribute name.
- Explicit description in DICOMScript - The private attribute VR can be explicitly defined in the DICOMScript. DVT will then use this VR to encode/decode the attribute values. **NOTE:** *The attribute name will not be known in this case.*

The Private Creator Codes (gggg,0010) to (gggg,00FF) must be defined by the User. DVT does not make any attempt to map Private Creator Codes - see below:

SEND C-STORE-RQ "CT Image" (

```
...
(0x00110010, LO, "Private Inc. - V1.1") # User responsible for correct Private Creator Code
(0x00111010, PN, "Secret Name")         # Attribute value defined with explicit VR
...
(0x00290022, LO, "Private Inc. - V1.2") # User responsible for correct Private Creator Code
(0x00292210, US, 0x1FE)                 # Attribute value defined with explicit VR
(0x00292211, US, 0x1FE)                 # Attribute value defined with explicit VR
...
)
```

7.11 Image (Object) Relationship Analysis

DVT will automatically produce a “directory” structure describing the detected relationship between objects stored during a Script or Emulator Session when the **Log Relation** flag is enabled.

The analysis shows the Patient/Study/Series/Object relationship between objects.

The **Patient ID (0010,0020)**, **Study Instance UID (0020,000D)**, **Series Instance UID (0020,000E)** and **SOP Instance UID (0008,0018)** are used in the relationship analysis.

In addition, DVT displays (when available) the **Patient Name (0010,0010)**, **Referenced Image Sequence (0008,1140)**, **Frame of Reference UID (0020,0052)** for each image, together with the third value of the **Image Type (0008,0008)** (e.g., for CT this defines the image as a LOCALIZER (Scout) or an AXIAL image).

***NOTE:** DVT does not interpret whether the relationship is correct, i.e. that intended by the SUT - that must be done by the user according to the Test Scenario.*

7.12 Replay Feature

DVT will display the received dataset attributes in ascending Tag order (in the detailed Results File). The format of the display (and the Results File) is such that the content of the dataset can be re-used in another DICOMScript (parsed by DVT).

Using an editor, simply cut the received dataset attribute values and paste them into an other DICOMScript. This feature is particularly useful when DVT (acting as an SCP) receives images from an SCU - the received image dataset can be copied for re-use in a SCU DICOMScript - the User can easily modify any of the image attribute values.

***NOTE:** Since DVT uses the backslash character (“\”) as a prefix for a HEX value (see section 7.7, “Extended Character Sets”), all backslashes in the pasted text must be changed to a double backslash (“\\”) to be correctly read by DVT. The exceptions to this are filenames and for the Image Display Format, DICOM tag (2010, 0010), which will allow use of a single backslash.*

8. Interpreting the validation results

8.1 Status Keywords

DVT uses seven kinds of “status” keywords:

- PASSED
- FAILED
- ERROR
- WARNING
- INFORMATION
- DEBUG
- RELATION
- Conditional Text(see sec 8.2)

8.1.1 PASSED

DVT uses the PASSED status as the final result of a successful validation. No ERRORs have been detected.

This is the goal for all DICOM conforming products!

8.1.2 FAILED

DVT uses the FAILED status as the final result of an unsuccessful validation - the total number of ERROR and/or WARNING messages is displayed.

8.1.3 ERROR

The ERROR status is used when validating DICOM Messages. Anything which is incorrect in the syntax or semantics of the DICOM Message will be reported as an ERROR.

Errors are prohibitive and occur when an implementation is incorrect. The cause of all ERRORs should be fixed before a SUT can be declared as conforming to DICOM.

The total number of ERRORs detected during the validation is displayed by DVT.

8.1.4 WARNING

The WARNING status is used when validating DICOM Messages. Anything which requires further attention, but is not incorrect, is reported as a WARNING - e.g., ACR-NEMA 2.0 Date/Time formats, unexpected Defined Terms, etc.,.

Warnings are not prohibitive. However, it may well be worth noting the cause of the warning for future reference.

The total number of WARNINGs detected during the validation is displayed by DVT.

8.1.5 INFO (INFORMATION)

The INFORMATION status is used when DVT wishes to simply inform the user of something - replacing Definitions, etc.

No user action is needed for INFORMATION status messages.

8.1.6 DEBUG

The DEBUG status is used to have DVT display detailed debugging information. Some of this low-level information may be useful to the User when difficult problems are encountered.

No user action is needed for DEBUG status messages.

8.1.7 RELATION

The RELATION keyword is used to mark the beginning of the Image(Object) Relationship Analysis.

8.2 Conditional Attribute Validation Output

DVT will validate if a conditional attribute should be present or not based on rules in the definition files. The output of this validation is described here. The basic form of the output is:

T[*condition*] – indicating that *condition* is true, or

F[*condition*] – indicating that *condition* is false.

condition can be one of the following:

PRESENT *tagAddress* – true if *tag* at *tagAddress* is present in the dataset

EMPTY *tagAddress* – true if *tag* at *tagAddress* is present in the dataset and has no value

VALUE *tagAddress value_number comparison_operator value* – true if the *value_number* value of *tag* at *tagAddress* meets the comparison with the given *value*. The *comparison_operators* are “<”, “<=”, “=”, “>=”, and “>”.

NOT T/F[*condition*] – true if *condition* is false

T/F[*condition1*] AND T/F[*condition2*] – true if *condition1* and *condition2* are true

T/F[*condition1*] OR T/F[*condition2*] – true if one or both of *condition1* or *condition2* are true

tagAddress can be composed of the following:

tag – the tag used in the condition is found at any level at or below the current level.

/tag – the tag used in the condition is found at the root level of the SOP instance.

./tag – the tag used in the condition is found at the same level as the tag to which the condition is applied.

../tag – the tag is one level up in the dataset
– i.e., from a nested item to the parent sequence level.

tag1/tag – the tag is one level down in the dataset
– i.e., tag1 is a sequence and the tag is to be found in the first item of the sequence.

../ and *tag1/* can be combined to produce a “path” to the attribute.

Example: *../0x00081100/0x00081050*

– the condition depends on the tag 0x00081050 at the relative location specified by *../0x00081100*.

DICOM has introduced the notion that a conditional attribute “may be present otherwise” or “may not be present otherwise”. To support the “maybe” option the concept of a WEAK condition has been introduced.

Example:

Clinical Trail Subject ID (0012,0040) shall be present if Clinical Trail Subject Reading ID is absent. May be present otherwise.

This is defined as **WEAK ((NOT PRESENT 0x00120042), TRUE)** in the Definition Files.

The condition is covered by **NOT PRESENT 0x00120042** and the “may be present” by **TRUE** (a secondary condition).

When the result is displayed, the first character, **T** or **F**, indicates the result of the overall condition evaluation. The rest of the display gives the reason for the result.

Some examples are:

T[PRESENT 0x00081120] – indicates that the condition is true because tag 0x00081120 is present in the dataset.

F[NOT T[PRESENT 0x00081120/0x00081050]] – indicates that the overall condition is false because tag 0x00081050 is present in the first item of the sequence 0x00081120 in the dataset. The condition in this case is “NOT PRESENT 0x00081120/0x00081050”. The validation results can be read from the “inside” of the expression as “tag 0x00081120/0x00081050 is present (T), the not of true is false”.

F[VALUE 0x00280002 1 > 1] – indicates that the overall condition is false because the first value of tag 0x00280002 is not greater than 1.

F[F[VALUE 0x00280004 1 = PALETTE COLOR] OR F[VALUE 0x00280004 1 = ARGB]] – indicates that the overall condition is false. The condition in English is “the first value of tag 0x00280004 is “PALETTE COLOR” or “ARGB””. In this case, the first value of tag 0x00280004 is not either of “PALETTE COLOR” or “ARGB”, so the overall condition is false (false or false = false).

8.3 Other Remarks

When interpreting the validation results generated by DVT, remember:

1. The Called AE Title and Calling AE Title in both the Associate Request and Associate Accept should be padded with trailing SPACE characters (to a total length for AE of 16). DVT automatically pads the values sent to the SUT. See [1] - part 8 - Table 9-11 - page 31 “... shall be encoded as **16 characters** as defined by the ISO 646:1990-Basic G0 Set with leading and trailing spaces (20H) being non-significant”.
2. UIDs should not be padded with a leading zero in any of the numeric components, e.g., “1.2.3.4” and “1.2.3.0” are valid UIDs whereas “1.2.3.04” is invalid. **NOTE:** If the OEM uses a date/timestamp to generate the UID, check that the full year YYYY is used and not YY (as in “97”) - this will be “00” at the turn of the century and generate an invalid UID (leading zeroes). See [1] - part 8 - F.1 ENCODING RULES - page 51 “... Leading 0’s of each component are not significant and **shall not be sent**”.
3. Type 1 attributes must be present in the Image with at least one value. The number of values is defined by the attribute VM (Value Multiplicity).
4. Type 2 attributes may be present with a zero value length, when the SCU does not know the corresponding attribute value.
5. When DVT will indicate an Attribute VR Format ERROR when the format of an attribute value is incorrect for its VR. See [1] - part 5 - Table 6.2-1 for the formats of all VR types defined.
6. DVT will indicate an ERROR when an attribute has a value that does not exactly match it’s possible Enumerated Values.
7. DVT will indicate a WARNING when an attribute has a value that does not exactly match it’s possible Defined Terms. If the OEM sends a value which is not one of the Defined Terms, the OEM should state this in the SUT Conformance Statement.
8. Superfluous attributes are those found as part of the Image, but which do not belong to the corresponding Definition - i.e., they are superfluous to the definition.
9. If extended characters are used in the Patient Information, the Specific Character Set attribute (0008,0005) must be present. Some OEMs may send a value “ISO-IR XXX” (**NOTE:** Hyphen in between ISO and IR) - this should be “ISO_IR XXX” (**NOTE:** Underscore).
10. Check that the Image(Object) Relationship Analysis produced by DVT matches that expected for the number of images received.

9. Advanced programming

DVT supports a number of DICOMScript commands that allow a more advanced and extensive validation to be performed. A full description of each command is given in the DICOMScript Language section later.

9.1 Data-warehouse

DVT includes a data-warehouse in which the User stores ACSE Requests & Responses, DICOM Commands, Dataset and Item objects (DICOMObjects) for re-use in the same or other DICOMScript.

The data-warehouse allows DICOMObjects to be shared between Test Sessions. This feature is particularly useful when performing a full system validation where more than one DICOM connection by be used (Example: A Modality can make use of the Basic Modality Worklist SOP Class, Storage SOP Class and Performed Procedure SOP Class. These can be handled within separate Test Sessions with the exchanged DICOMObjects being shared between sessions.)

9.1.1 Create, Set and Delete

The data-warehouse can be filled by creating ACSEObjects and DICOMObjects within the DICOMScript using the **CREATE** command.

The ACSEObjects will be automatically identified by DVT and can only be used within a single DICOMScript.

CREATE ASSOCIATE-RQ
CREATE ASSOICATE-AC
CREATE ASSOICATE-RJ
CREATE RELEASE-RQ
CREATE RELEASE-RP
CREATE ABORT-RQ

***NOTE:** The format of the ACSEObject identifier is IDnnnn when nnnn is a random number generated by DVT when executing the DICOMScript. The use of these identifiers is purely for backwards compatibility.*

Each DICOMObject must be identified in the data-warehouse.

***NOTE:** Take care when naming the identifiers. The names should be unique.*

CREATE <command_ref> # Create a DICOM Command object
CREATE <command_ref> <dataset_ref> # Create a DICOM Command / IOD combination
CREATE <item_ref> [DEFINED_LENGTH] # Create an Item

DEFINED_LENGTH is used when creating Items to indicate that the item should be encoded with a defined length for export. This option can be used to over-rule the default export item encoding which uses the undefined length format.

It is also possible to create Media Storage objects in the data-warehouse.

CREATE FILE-HEAD
CREATE FILE-TAIL

Once an ACSEObject or DICOMObject has been created in the data-warehouse, the paramater/attribute values can be defined using the **SET** command. Any number of SET commands can be applied to an ACSEObject or DICOMObject. The actual attribute values used will depend on the values given at the time the ACSEObject or DICOMObject is used.

The syntax for the ACSEObject **SET** commands is the same as that described above for the SEND/RECEIVE commands.

SET <command_ref> <dicom_object_attribute_list>
SET <dataset_ref> <dicom_object_attribute_list>
SET <item_ref> <dicom_object_attribute_list>

Media Storage objects can be set as well.

SET FILE-HEAD <file_head_param_list>

```
<file_head_param_list> (
  PREAMBLE <integer>           # Preamble value.
  PREFIX <string 4>             # Defaults to "DICM".
  TRANSFER-SYNTAX <uid>        # Of Stored Dataset.
)
```

SET FILE-TAIL <file_tail_param_list>

```
<file_tail_param_list> (
  DATASET-TRAILING-PADDING <YES/NO> # Indicator if trailing padding should be added.
  PADDING-VALUE <integer>           # Value used for padding.
  SECTOR-SIZE <integer>             # Sector size when producing trailing padding.
)
```

An ACSEObject or DICOMObject can be removed from the data-warehouse using the **DELETE** command.

DELETE ASSOCIATE-RQ
DELETE ASSOICATE-AC
DELETE ASSOICATE-RJ
DELETE RELEASE-RQ
DELETE RELEASE-RP
DELETE ABORT-RQ

DELETE <command_ref>

DELETE <dataset_ref>

DELETE <item_ref>

Media Storage objects can be removed from the data-warehouse too.

DELETE FILE-HEAD

DELETE FILE-TAIL

Example 1:

CREATE N-GET-RSP NGETRSPID

Create an N-GET-RSP DICOMObject in the data-warehouse and identify it as "NGETRSPID".

Example 2:

CREATE C-STORE-RQ CSTORERQID "CT Image" CTIMAGE1

Create a C-STORE-RQ and "CT Image" DICOMObject in the data-warehouse. The C-STORE-RQ DICOMObject will be identified as "CSTORERQID" and the "CT Image" DICOMObject will be identified as "CTIMAGE1".

Example 3:

CREATE "Source Image Sequence – Item" ITEMID1

Create an item in the data-warehouse representing a Source Image Sequence item and identify it as "ITEMID1".

Example 4:

```

SET "CT Image" CTIMAGE1 (
...
(0x00100010, "Doe^John")
...
(0x00080008, "ORIGINAL", "PRIMARY", "LOCALIZER")
(0x00082112, SQ, "ITEMID1")
...
(0x00280004, "MONOCHROME1")
(0x00280010, 0x200)
(0x00280011, 0x200)
...
(0x7FE00010, OB, "ct-img.dat")
...
)

```

Set the CT image DICOMObject with identifier "CTIMAGE1" with the attribute values defined. The Source Image Sequence item is defined by reference (using the item identifier of the previously created "Source Image Sequence – Item" ITEMID1 in example 3).

The syntax of the SET when defining attributes and values is exactly the same as for the SEND / RECEIVE commands.

Example 5:

DELETE C-STORE-RQ CSTORERQID

Delete the C-STORE-RQ DICOMObject with identifier "CSTORERQID" from the data-warehouse.

Example 6:

DELETE "CT Image" CTIMAGE1

Delete the CT image DICOMObject with identifier "CTIMAGE1" from the data-warehouse.

9.1.2 Import, Export

Another way in which DICOMObjects can be stored in the data-warehouse is by importing them over an existing Association (network connection) using the **IMPORT** command. The DICOMObject being imported needs to be identified for later use.

IMPORT <dicom_message>

```

<dicom_message>      <command_ref>
                      | <command_ref> <dataset_ref>

```

DICOMObjects in the data-warehouse can be exported over an existing Association (network connection) using the **EXPORT** command. The Transfer Syntax agreed for the DICOMObject SOP Class will be used when exporting the DICOMObject.

EXPORT <dicom_message>

```

<dicom_message>      <command_ref>
                      | <command_ref> <dataset_ref> [ <pres_context_id> ]

```

```

<pres_context_id>    PRESENTATION-CONTEXT-ITEMS( <integer> )
                      - option to force the given Presentation Context ID to be used.

```

Example 1:

IMPORT C-STORE-RQ “MR Image” MRIMAGE1

Import an MR image DICOMObject into the data-warehouse with identifier “MRIMAGE1”. The import command will automatically create the object at the time of import.

Example 2:**EXPORT C-STORE-RQ “MR Image” MRIMAGE2**

Export the MR image identified by “MRIMAGE2” over the existing Association. The MR image must exist in the data-warehouse to allow the export to be done.

9.1.3 Read, Write

The final way in which DICOMObjects can be stored in the data-warehouse is by reading them from Media Storage files using the **READ** command. The DICOMObject being read needs to be identified for later use.

READ *filename* <dataset_ref>

or

READ *filename* <tag>

The value of the attribute with <tag> will be extracted from the DICOMObject during the read. This attribute value will then be used as the identifier for this DICOMObject. This allows reference objects to be stored in the data-warehouse and accessed by the value of a given tag. It is possible, for example, to use the SOP Instance UID (0008,0018) as the tag – all DICOMObjects read will then be stored using their SOP Instance UID as the identifier. During Storage SCP Emulation and Media Validation, the data-warehouse is searched for DICOMObjects with a matching <tag>. In this case it would be possible to have the DICOMObjects received by the Emulator or read by the Media validated against matching reference objects. This is useful to ensure that no changes have been made in DICOMObjects used.

DICOMObjects in the data-warehouse can be written into a Media Storage file using the **WRITE** command. The Transfer Syntax defined in the FILE-HEAD DICOMObject is used when writing the DICOMObject.

WRITE *filename* <dataset_ref>

If the full pathname is not provided in *filename*, the path will be relative to the script file directory for the **WRITE** command. For the **READ** command, the file will be looked for relative to the script file directory and, if it is not there, then the file will be looked for relative to the session file directory.

Example 1:**READ “ctimage.dcm” “CT Image” CTIMAGE1**

Read the Media Storage file ctimage.dcm and store the extracted DICOMObject in the data-warehouse with identifier “CTIMAGE1”.

Example 2:

READ “ctimage1.dcm” 0x00080018

READ “ctimage2.dcm” 0x00080018

READ “mrmimage1.dcm” 0x00080018

Read the Media Storage files ctimage1.dcm, ctimage2.dcm and mrmimage1.dcm. Extract the attribute value for tag 0x00080018 from each image and use the value as the identifier when storing the DICOMObjects in the data-warehouse.

NOTE: When using this form of the **READ** command it is important that all files read use the same tag value as the identifier. The Storage SCP Emulator and Media use the value of the last tag read as the search criteria for reference DICOMObjects in the data-warehouse.

Example 3:

WRITE “mrImage.dcm” “MR Image” MRIMAGE2

Write the MR image DICOMObject identified as “MRIMAGE2” into the file named mrImage.dcm.

9.2 Reuse of objects and attributes

Once DICOMObjects are stored in the data-warehouse, they can be manipulated using some of the DICOMScript commands described below.

9.2.1 Compare

The **COMPARE** command compares two attributes identified by their tag values. These tags may be different attributes in the same DICOMObject or in different DICOMObjects.

The comparison is made as follows:

1. Check for presence of the attributes in both DICOMObjects. Attributes must be present in both DICOMObjects.
2. Compare the VR and VM of the attributes in both DICOMObjects. VR and VM must be the same for both attributes.
3. Compare the attribute values in both DICOMObjects. The attribute values must be the same. Insignificant leading and trailing padding will be ignored during the comparison.

The DICOMScript will fail when COMPARE encounters a difference. COMPARE_NOT will succeed when the given tags are not equal.

COMPARE <dicom_object_ref> <tag> <dicom_object_ref> <tag>

COMPARE_NOT <dicom_object_ref> <tag> <dicom_object_ref> <tag>

NOTE1: One or both tags not present in the DICOMObjects is interpreted as them being not equal.

NOTE2: Two different <tag> values can be compared in the same <dicom_object_ref> or the same <tag> can be compared in two different <dicom_object_ref>s.

Example 1:

COMPARE “SC Image” IMAGE1 0x7FE00010 “SC Image” IMAGE2 0x7FE00010

The comparison operation is successful if the Pixel Data in the two Secondary Capture images is the same.

Example 2:

COMPARE_NOT “SC Image” IMAGE1 0x7FE00010 “SC Image” IMAGE2 0x7FE00010

The comparison operation is successful if the Pixel Data in the two Secondary Capture images is different.

9.2.2 Confirm

The **CONFIRM** command is used in conjunction with the ECHO command to enable the User to confirm that the test may be continued now that the required actions have been done. DVT will display a message box when the confirmation is required.

CONFIRM

Example:

ECHO “Please send CT Image number 1.”

ECHO “Hit <ENTER> when ready...”

CONFIRM

9.2.3 Copy

The **COPY** command will copy attributes. This can be between two different DICOMObjects or two different attributes within one DICOMObject.

Copying a non-existent source attribute will remove the destination attribute (if it was present before the copy). The destination attribute will have the VR, VM and value(s) set equal to the source attribute.

COPY <destination dicom_object_ref> <tag> <source dicom_object_ref> <tag>

The first object / attribute combination is the COPY destination. By defining the same <dicom_object_ref> twice with different <tag> numbers, an attribute is copied within one DICOMObject.

NOTE: The copy command can add attributes that conflict with the definitions in the Definition Files (e.g., other VR, unexpected attribute, etc).

Example:

COPY “SC Image” IMAGE1 0x7FE00010 “SC Image” IMAGE2 0x7FE00010

The examples below will copy the Pixel Data of standard secondary capture image object “IMAGE2” to the destination object “IMAGE1”.

9.2.4 Delay

The **DELAY** command is used to introduce a delay, in seconds, in the execution of the DICOMScript.

DELAY <integer>

NOTE: It is not possible to interrupt a DICOMScript which as been delayed (the delay is implemented by putting the script thread to sleep for the delay period).

Example:

DELAY 10

9.2.5 Display

The **DISPLAY** command can either display an attribute selection of a DICOMObject or all attributes present in the DICOMObject.

DISPLAY <dicom_object_ref> [<tag> [<tag>]...]

Example 1:

DISPLAY “CT Image” IMAGE1 0x00080008 0x00080016

Display an attribute selection (two attributes).

Example 2:

DISPLAY “CT Image” IMAGE1

Display the whole object.

9.2.6 Echo

The **ECHO** command is used to display messages to the User. The messages may just be of an informative nature or provide the User with instructions on how to continue with the test scenario.

ECHO <string 256>

Example:

ECHO “This test scenario will form an Association with the Tested System,”

ECHO “requesting support for CT Image storage.”

9.2.7 Reset

The **RESET** command can be used to:

- Reset the content of the Data Warehouse (remove all entries).
- Reset the relationship analysis/stored instance list (used by the Storage Commitment SCP Emulator).
- Reset the state of the DULP Finite State Machine (to allow new Associations – Presentation Context ID is reset to 1).
- Reset the Script Execution Context to the current Session Property values.

The command takes one of the following forms:

RESET ALL – reset Data Warehouse content, Relationship/Stored Instance List, DULP association state and Script Execution Context.

RESET WAREHOUSE – reset Data Warehouse content only.

RESET RELATION – reset Relationship/Stored Instance List only.

RESET ASSOCIATION – reset DULP association state only.

RESET SCRIPT-EXECUTION-CONTEXT – reset the Script Execution Context only.

9.2.8 System

The **SYSTEM** command allows any operating system call to be made from the DICOMScript. The string given as an argument to the **SYSTEM** command is passed directly to the command shell for interpretation.

The command line version of DVT can be started from within a DICOMScript. This is useful when testing the MOVE service between three devices - DVT acting as MOVE SCP (and via second process STORE SCU).

***NOTE:** Be aware that system commands depend on the operating system (Unix or Windows).*

SYSTEM <string 256>

Example:

SYSTEM “command to execute”

9.2.9 Time

The **TIME** command is used to print timestamps in the Display Pane. Successive calls will also give a delta with the previous **TIME** call.

TIME

9.2.10 Validate

The **VALIDATE** command is used to validate DICOMObjects stored in the data-warehouse.

The first step is to validate the DICOMObject against the definitions in the Definition File. If the Definition File is system specific (not the general DICOM.Definition), the first step ensures that the DICOMObject conforms to the DICOM Standard and, in particular, to the SUT’s Conformance Statement.

As a second step, the DICOMObject can be compared against an optional list of reference DICOMObjects.

VALIDATE <dicom_object> <reference_dicom_object_list>

<reference_dicom_object_list> <dicom_object> [OR <dicom_object> ...]

Example 1:

VALIDATE C-STORE-RQ STORE1 “XA Image” IMAGE1

Validation of a Standard XA image against the Definition File only.

Example 2:

VALIDATE C-STORE-RQ STORE1 “XA Image” IMAGE1

C-STORE-RQ REFSTORE1 “XA Image” REFIMAGE1

Validation against the Definition File and a reference DICOMObject.

9.2.11 Verbose

The **VERBOSE** command is used to enable / disable the Session “verboseness” from within the DICOMScript. The **VERBOSE ON / OFF** command allows important sections of the DICOMScript to be highlighted (verbose output produced), avoiding unnecessarily long verbose output of the whole DICOMScript. The condition of the **VERBOSE ON / OFF** state is maintained even after the script finishes execution. If a script exits with **VERBOSE OFF**, normal output will not resume for that session until a script is run with a **VERBOSE ON** command.

***NOTE:** The **VERBOSE** command is disabled when the **LOG_DEBUG** option is used so that the full DICOMScript can be seen – for de-bugging purposes.*

VERBOSE ON

VERBOSE OFF

Example 1:

The Verbose works like a toggle of the verbose option passed to DVT:

VERBOSE ON

Interesting Section - DICOMScript content will be displayed according to the Session “verboseness”.

VERBOSE OFF

Remainder of DICOMScript content displayed silently (no verbose output).

9.3 Script Execution Context

It is possible to modify some Test Session Properties during the DICOMScript execution from within the DICOMScript itself – the values defined in the DICOMScript are copied into what is known as the Script Execution Context. The following commands can be used to modify the Script Execution Context.

9.3.1 Add Group Length

The **ADD-GROUP-LENGTH** command will overrule the current Test Session Property **ADD-GROUP-LENGTH** in the Script Execution Context. The value defined remains active until the end of the script execution or until modified by another **ADD-GROUP-LENGTH** command.

ADD-GROUP-LENGTH ON

ADD-GROUP-LENGTH OFF

Example:

ADD-GROUP-LENGTH ON

The DICOM objects to be sent, following this command, will all be encoded by DVT with Group Length attributes for all groups in the objects.

9.3.2 Application Entity

The **APPLICATION-ENTITY** command is used to define the Definition File AE Name and AE Version to use for the validation of any DICOM objects received by DVT, after this command, in the DICOMScript. This provides a mechanism to switch between Definition Files during the script execution.

APPLICATION-ENTITY <ae_name> <ae_version>

<ae_name> <string 256>

<ae_version> <string 256>

Example:

APPLICATION-ENTITY "OEM SUT" "V1.0"

VALIDATE C-STORE-RQ STORE1 "XA Image" IMAGE1

APPLICATION-ENTITY "DICOM" "3.0"

VALIDATE C-STORE-RQ STORE2 "XA Image" IMAGE2

The XA Image IMAGE1 will be validated against the SUT definition for OEM V1.0. The XA Image IMAGE2 will be validated against the DICOM 3.0 definition.

9.3.3 Define Sequence Length

The **DEFINE-SQ-LENGTH** command will overrule the current Test Session Property **DEFINE-SQ-LENGTH** in the Script Execution Context. The value defined remains active until the end of the script execution or until modified by another **DEFINE-SQ-LENGTH** command.

DEFINE-SQ-LENGTH ON

DEFINE-SQ-LENGTH OFF

Example:

DEFINE-SQ-LENGTH ON

The DICOM objects to be sent, following this command, will all be encoded by DVT with Defined Sequence Lengths for all Sequences in the objects.

9.3.4 Populate

The **POPULATE** command is used to enable / disable the automatic addition of Type 2 attributes during DICOMObject encoding. This ensures that DICOMObjects used by DVT will conform to the DICOM Standard without the User having to explicitly define values for all Type 2 attributes.

POPULATE ON

POPULATE OFF

Example:

```
CREATE C-STORE-RQ STORE1
POPULATE OFF
CREATE C-STORE-RQ STORE2
POPULATE ON
CREATE C-STORE-RQ STORE3
```

Three DICOMObjects are created, the first DICOMObject “STORE1” will be populated on encoding as this is the default setting. “STORE2” will not be populated. “STORE3” will be populated.

9.3.5 Strict Validation

The **STRICT-VALIDATION** command will overrule the current Test Session Property **STRICT-VALIDATION** in the Script Execution Context. The value defined remains active until the end of the script execution or until modified by another **STRICT-VALIDATION** command.

```
STRICT-VALIDATION ON
STRICT-VALIDATION OFF
```

Example:

```
STRICT-VALIDATION ON
```

Strict Validation of any DICOM objects will now be performed by DVT.

9.3.6 Validation

The **VALIDATION** command defines how the following DICOM objects will be validated by DVT. The following options are supported:

- | | |
|----------------------------------|--|
| ◆ ENABLED | - Validate the following DICOM Objects as normal – full validation. |
| ◆ ENABLED-USE-DEF-ONLY | - Validate the following DICOM Objects against the Definition File only. |
| ◆ ENABLED-USE-VR_ONLY | - Validate the following DICOM Objects for the VR syntax of the attributes only. |
| ◆ ENABLED-USE-REF-ONLY | - Validate the following DICOM Objects against any corresponding reference object in the DICOMScript only. |
| ◆ ENABLED-USE-DEF-AND-VR | - Validate using Definition and VR only. |
| ◆ ENABLED-USE-DEF-AND-REF | - Validate using Definition and Reference only. |
| ◆ ENABLED-USE-VR-AND-REF | - Validate using VR and Reference only. |
| ◆ DISABLED | - Do not validate DICOM Objects. |

```
VALIDATION ENABLED
VALIDATION ENABLED-USE-DEF-ONLY
VALIDATION ENABLED-USE-VR_ONLY
VALIDATION ENABLED-USE-REF-ONLY
VALIDATION ENABLED-USE-DEF-AND-VR
VALIDATION ENABLED-USE-DEF-AND-REF
VALIDATION ENABLED-USE-VR-AND-REF
VALIDATION DISABLED
```

10. Visual Basic Scripts

DVT (in this chapter, DVT refers to the GUI and command line version) offers the possibility to execute Visual Basic Scripts. These scripts may use calls from the Dvtk and DvtkData library that are supplied with DVT to implement the same functionality as may be implemented by Dicom Scripts. Furthermore, Visual Basic Script contains more language constructs like e.g. the “If” statement, that may be used to implement conditional program flow (this is not possible with Dicom Scripts).

10.1 Entry point

The entry point for a Visual Basic Script executing under DVT GUI is the procedure Main contained in the module DvtkScript. The Visual Basic Script must contain at least the following lines:

```
Module DvtkScript
    Sub Main()
        ... ` Entry point of the code
    End Sub
End Module
```

The entry point for a Visual Basic Script executing under the command line version of DVT is the procedure DvtCmdMain contained in the module DvtkScript. The Visual Basic Script must contain at least the following lines:

```
Module DvtkScript
    Sub DvtCmdMain(ByVal CmdArgs() As String)
        ... ` Entry point of the code
    End Sub
End Module
```

So a Visual Basic Script that must work with both the DVT GUI and the command line version of DVT must implement both entry points. By having two separate entry points, it is now possible to implement slightly different behavior between a Visual Basic Script executed by the DVT GUI and the command line version. E.g. when executed from the DVT GUI a popup dialog requesting user input may be displayed but when executed from the command line version no popup is displayed and default values are used. The command line version that is typically used in batch processing can now be executed without user interaction.

10.2 .NET interface Dvtk and DvtkData

The .NET interface for the Dvtk and DvtkData library is available in a Visual Basic Script executed by DVT. The complete .NET interface of these libraries is described in the following two files in the <INSTALLDIR>\docs directory:

- DvtkDocumentation.chm: compiled HTML help file for the Dvtk library.
- DvtkDataDocumentation.chm: compiled HTML help file for the DvtkData library.

10.3 Session variable

When executing a Visual Basic Script under DVT, the global “session” variable (pointing to the ScriptSession object the script belongs to) is available in the complete scope of the Visual Basic Script. Use this variable instead of creating your own Session object using the Dvtk.Sessions.ScriptSession.LoadFromFile method, because unsaved changes in the ScriptSession object will only be visible using the “session” variable. Also the StartResultsGathering and EndResultsGathering methods don’t have to be called on the ScriptSession object in the Visual Basic Script because DVT takes care of that. Finally, don’t call Dvtk.Setup.Initialize() or Dvtk.Setup.Terminate() because DVT also takes care of this.

10.4 Visual Studio .NET 2005

Developing Visual Basic Scripts for DVT is easier when using a development tool like Visual Studio .NET. The following are important steps to do this in Visual Studio .NET 2005:

- Create a new project.
- Add reference to the Dvtk.dll and DvtkData.dll. The complete .NET interface of the Dvtk and DvtkData libraries are now available.
- Call Dvtk.Setup.Initialize() first before using other calls.
- Call Dvtk.Setup.Terminate() as the last executed line of code.
- Use the Dvtk.Sessions.ScriptSession.LoadFromFile method to create a Session object, supplying it a Session file name.
- Use the Dvtk.Sessions.Session.StartResultsGathering and the Dvtk.Sessions.Session.EndResultsGathering methods on the Session object to start and end logging of the Results to the Xml file(s).
- Use other methods in combination with the Session object to do the actual testing.

11. Using Secure Sockets

11.1 Overview

To use secure sockets to communicate over TCP/IP to the SUT, the following steps must be taken.

1. The session must be configured to use secure sockets. This is done in the **Security Settings** dialog. Note that the DICOM standard port for secure sockets is port 2762.
2. The security credentials for DVT need to be generated. This will normally be done using the key generation tool in use by the organization. Otherwise, the certificate generation tool in DVT can be used to generate the credentials. The DVT generation tool is at **Security Settings - Keys - Create**. DVT also includes several sample credentials files that can be used.
3. The security credentials need to be imported into DVT using **Security Settings - Keys - Files - File containing DVT Private Keys**. Use the **Browse** button to import the credentials. Note that the order of the entries in this file is significant. See section 11.4.1, “Security Credentials File Format” for details.
4. The certificate for the SUT must be obtained or generated. DVT can be used to generate certificates and credentials for the SUT, or the sample credential and certificate files supplied with DVT can be used.
5. A certificate that is in the certificate chain of the SUT’s certificate must be imported into DVT as a trusted certificate using **Security Settings - Keys - Files - File containing SUT Public Keys**. Typically the root self-signed certificate is used as the trusted certificate. Use the **Browse** button to import the certificate or certificate chain.
6. A certificate that is in the certificate chain of DVT’s certificate must be added to the SUT’s trusted certificate list.
7. The SUT must be configured to use secure sockets.
8. The SUT can now be tested as normal. No changes to the scripts are needed. Secure sockets will be used automatically.

The secure socket configuration is done on a per session basis. The same Security Credentials file and Trusted Certificate file can be used by more than one session.

By default, the Security Credentials file and Trusted Certificate file are stored in the same directory as the session file.

In a normal application, the private key must be kept secret and secure. So it must be encrypted as it is stored on the hard disk. DVT allows the use of password protected encrypted private keys. DVT will ask for the password when the session is opened, or when the use of secure sockets is turned on, or when the security credentials file is selected. The User can cancel entering the password and continue to use the session, but any attempt to open a secure socket connection will fail. DVT also has a **default password** that is unknown to the User. This allows DVT to open the session without asking the User for a password and is slightly more secure than using unencrypted private keys. It is not truly secure, since anyone with access to DVT can access the private key. DVT also supports unencrypted private keys by leaving the password blank.

DVT displays the certificate valid times using UTC time.

Turning on Debug logging will show details about the startup of the secure connection.

11.2 Some Definitions

The following definitions are helpful in understanding the use of secure sockets.

Certificate - A digitally signed document that identifies an entity and provides that entity’s public key. Every certificate is traceable to a self-signed certificate through its certificate chain.

Certificate Chain - A list of certificates that starts at an entity’s certificate and includes the certificate of the signer of the certificate. If the signer’s certificate is signed, then that signer’s certificate is also included. Each signing certificate is included until a self-signed certificate is reached.

Private Key - Used in Public Key Encryption where there is a private key that is held in secret by one of the parties and a public key that anyone can know. Data that is encrypted with one of the keys can only be decrypted using the other key.

Public Key - Used in Public Key Encryption where there is a private key that is held in secret by one of the parties and a public key that anyone can know. Data that is encrypted with one of the keys can only be decrypted using the other key.

Security Credentials - Security credentials are the private key and certificate chain that represent DVT.

Self-Signed Certificate - A certificate that is signed by the owner of the certificate. A self-signed certificate can also be called a root certificate.

Trusted Certificate - A certificate that is known by DVT to be valid. Typically, trusted certificates are self-signed certificates. If the trusted certificate is not self-signed, then the certificate's certificate chain must be included in the trusted certificate file.

11.3 Overview of TLS/SSL

TLS is a widely used standard security protocol. TLS is an additional layer added right above TCP/IP. This allows the main part of an application to have no changes when switching to secure sockets.

TLS is a framework that puts together other security methods to provide a secure transport package. For each of the functions, TLS supports many security methods. Which methods are used is negotiated at connection time.

The TLS negotiation is initiated after the socket is opened but before any data is sent.

The TLS Handshake Protocol involves the following steps (this does not represent the actual message exchange):

1. The client provides a list of possible protocols and mechanisms that can be used for the session to the server. The client can also request that a session be reopened.
2. The server selects the protocol and mechanisms to be used from the options provided by the client and sends these back to the client. If the session is being resumed, the negotiation is complete.
3. The server uses the entity authentication method selected to send its certificate chain to the client. The certificate chain includes the server's public key and all of the signatures back to a Certificate Authority that are used to verify that the public key belongs to the server.
4. The client verifies the server certificate.
5. The client sends its certificate chain if requested by the server.
6. The server verifies the client certificate if sent.
7. The server uses the key exchange method to send key information to the client.
8. The client sends key information to the server to complete the key exchange. The result is a key to be used for symmetric encryption.
9. Negotiation is completed.

Any failures during the negotiation result in the socket being closed.

After the negotiation is complete, data can be sent over the socket. If encryption was agreed to, the data will be encrypted using the symmetric key algorithm and key that was negotiated for the session.

11.4 File Formats

All security files used by DVT are stored in Base64 encoded PEM format. In this format each of the entries in the file are bracketed by BEGIN and END lines. The data is in an ASCII representation of binary data. This allows the ordering and contents of the files to be manipulated with a standard text editor.

DVT uses the file extension of **.pem** for the files. Many other systems (including Windows) may require that an extension of **.cer** be used. A DVT PEM certificate may be viewed using the Microsoft Windows utilities by changing the file extension to .cer and double-clicking the file. Only the first certificate in the file can be viewed in this manner.

DVT can import certificates in the following formats: Base64 PEM, DER, PKCS #7, PKCS #12.

11.4.1 Security Credentials File Format

A security credentials file contains DVT's private key(s) and certificate chain(s). The structure of the file is:

1. Private key

2. Certificate that corresponds with the private key
3. Certificate of the signer of the first certificate
4. Certificate of the signer of the previous certificate until the self-signed certificate is reached
5. Repeat the above for the other type of private key, if present

DVT supports 2 types of keys: RSA and DSA. The type of key used must correspond to the Authentication method defined in the ASCE Properties dialog box. DVT can be configured to allow both types at the same time.

Note that it is valid to sign one type of key with the other type of key.

11.4.2 Trusted Certificate File Format

The trusted certificate file contains the certificate chains of the certificates that DVT considers trusted. The order of the certificate chains in the file is not significant. Each certificate chain starts with a certificate that has not signed another certificate in the file. It is then followed by the certificate that signed it, and so on, until the self-signed certificate for the chain is reached.

11.4.3 Generated Certificate and Private Key File Formats

The output from the Generate Certificate dialog box is two files. The first file contains the certificate chain for the generated certificate. The generated certificate is the first certificate in the chain. The other file contains only the generated private key. Note that this is not the credentials file. See the next section for instructions on creating the credentials file from the output files.

11.5 Supported Cipher Suites

The following table lists the set of Cipher Suites that are supported by DVT. The Cipher Suites are selected in the **Security Settings** dialog. The set of available suites is shared with the SUT when a secure socket connection is opened. The most secure Cipher Suite is chosen for the connection. The table below is ordered from the most secure to the least secure, so given a choice the entry closest to the start of the table will be used.

OpenSSL Cipher Name	Authentication Method	Key Exchange Method	Data Integrity Method	Privacy Method
DHE-RSA-AES256-SHA	RSA	DH	SHA1	AES 256 bit key
DHE-DSS-AES256-SHA	DSA	DH	SHA1	AES 256 bit key
AES256-SHA	RSA	RSA	SHA1	AES 256 bit key
EDH-RSA-DES-CBC3-SHA	RSA	DH	SHA1	Triple DES
EDH-DSS-DES-CBC3-SHA	DSA	DH	SHA1	Triple DES
DES-CBC3-SHA	RSA	RSA	SHA1	Triple DES
DHE-RSA-AES128-SHA	RSA	DH	SHA1	AES 128 bit key
DHE-DSS-AES128-SHA	DSA	DH	SHA1	AES 128 bit key
AES128-SHA	RSA	RSA	SHA1	AES 128 bit key
NULL-SHA	RSA	RSA	SHA1	None
NULL-MD5	RSA	RSA	MD5	None

11.6 Sample Certificate and Credentials Files

DVT comes with several sample credentials and certificates. They are stored in the directory <INSTALLDIR>\certificates. There are two self-signed entities, **Dvt** and **Sut**. For each entity, there is a credentials and a certificate file. The files are listed in the table below. All of the private keys in the files are unencrypted – the default password for these files is **dvt**.

Filename	Entity	Certificate or Credential?	Signed by
DvtSelfSigned.cer	Dvt	Certificate	self
DvtSelfSigned.p12	Dvt	Credentials	self
DvtSelfSignedCert.pem	Dvt	Certificate	self

Filename	Entity	Certificate or Credential?	Signed by
DvtSelfSignedCred.pem	Dvt	Credentials	self
SutSelfSigned.cer	Sut	Certificate	self
SutSelfSigned.p12	Sut	Credentials	self
SutSelfSignedCert.pem	Sut	Certificate	self
SutSelfSignedCred.pem	Sut	Credentials	self

12. Examples

A number of example DICOM(Super)Scripts are provided, as part of the DVT release package, to show the sort of tests that can be performed for a number of SOP classes. The examples are arranged in such a way as to allow DVT to act as both SCU and SCP, i.e., the SCU DICOM(Super)Scripts can be used against the SCP DICOM(Super)Scripts. The new User is advised to spend some time going through the examples in order to become more familiar with some of the underlying concepts.

NOTE: *The DICOM(Super)Scripts released as part of the DVT package only form a small subset of those needed to perform a thorough validation of a SUT. The User is encouraged to construct a set of DICOM(Super)Scripts relevant to the SUT being tested.*

Examples for the following SOP classes are provided:

- Storage SOP Classes
- Query SOP Classes
- Retrieve SOP Classes
- Modality Worklist Management SOP Class
- Modality Performed Procedure Step SOP Class
- Storage Commitment SOP Class
- Structured Reporting SOP Class
- Print Management Meta SOP Classes
- Verification SOP Class

When using the GUI version, open the Session File using menu option *File - Open* in the appropriate directory, and execute the DICOM(Super)Scripts as normal. Two parallel sessions can be opened – one as SCP and the other as SCU.

When running DVT SCU against DVT SCP always ensure that the DVT SCP is started first, thereby allowing a TCP/IP listen socket to be created for the DVT SCU connect socket to communicate with.

12.1 Storage SOP Class

Table 14-1 describes the example Storage SOP Class DICOM(Super)Scripts.

TABLE: 14-1 Storage SOP Class DICOM(Super)Scripts				
SCU Session		SCP Session		Scenario
My Documents\DVTK\Projects\Examples\scripts\storage\scu Session File: test001.ses		My Documents\DVTK\Projects\Examples\scripts\storage\scp Session File: test001.ses		
File Pane DICOM(Super)Script	Results File	File Pane DICOM(Super)Script	Results File	
1.ds	Summary_001_1_ds_res.xml	1.ds	Summary_001_1_ds_res.xml	Simulate Storage SOP Class not supported by SCP during Association - <i>“abstract syntax not supported (provider rejection)”</i>
2.dss	Summary_001_2_dss_res.xml	2.dss	Summary_001_2_dss_res.xml	Show some of the ERROR/WARNING messages produced by DVT when received image does not conform to the given SOP Class definition.
3.ds	Summary_001_3_ds_res.xml	3.ds	Summary_001_3_ds_res.xml	Simulate <i>“Out of Resources - A700”</i> by SCP on attempted image storage.
4.dss	Summary_001_4_dss_res.xml	4.dss	Summary_001_4_dss_res.xml	Send 6 images belonging to the same patient in one study and three series over a single Association. Show Image(Object) Relationship Analysis produced by SCP. See patient <i>One^Secondary Capture Image</i> in the Query examples.
5.dss	Summary_001_5_dss_res.xml	5.dss	Summary_001_5_dss_res.xml	Send 4 images belonging to the same patient in one study and one series. Each image is sent in a separate Association. Show Image(Object) Relationship Analysis produced by SCP. See patient <i>Two^Secondary Capture Image</i> in the Query examples.
6.dss	Summary_001_6_dss_res.xml	6.dss	Summary_001_6_dss_res.xml	Send 5 images belonging to the same patient in one study and two series. Each series is sent in a separate Association. Show Image(Object) Relationship Analysis produced by SCP. See patient <i>Three^Secondary Capture Image</i> in the Query examples.

12.2 Query SOP Class

Table 14-2 describes the example Query SOP Class DICOM(Super)Scripts.

TABLE: 14-2 Query SOP Class DICOM(Super)Scripts				
SCU Session		SCP Session		Scenario
My Documents\DVTK\Projects\Examples \scripts\query\scu Session File: test001.ses		My Documents\DVTK\Projects\Examples \scripts\query\scp Session File: test001.ses		
File Pane DICOM(Super)Script	Results File	File Pane DICOM(Super)Script	Results File	
1.ds	Summary_001_1_ ds_res.xml	1.ds	Summary_001_1_ ds_res.xml	Simulate SCP rejecting Association due to “Called AE Title not recognized”.
2.ds	Summary_001_2_ ds_res.xml	2.ds	Summary_001_2_ ds_res.xml	Perform query for all patients with surname beginning with “O”. Recurse until all Study, Series and Image information has been returned.

12.3 Retrieve SOP Class

Table 14-3 describes the example Retrieve SOP Class DICOM(Super)Scripts.

TABLE: 14-3 Retrieve SOP Class DICOM(Super)Scripts				
SCU Session		SCP Session		Scenario
My Documents\DVTK\Projects\Examples \scripts\retrieve\scu Session File: test001.ses		My Documents\DVTK\Projects\Examples \scripts\retrieve\scp Session File: test001.ses		
File Pane DICOM(Super)Script	Results File	File Pane DICOM(Super)Script	Results File	
1.ds	Summary_001_1_ ds_res.xml	1.ds	Summary_001_1_ ds_res.xml	Simulate SCP rejecting Association due to “Calling AE Title not recognized”.
2.ds	Summary_001_2_ ds_res.xml	2.ds	Summary_001_2_ ds_res.xml	Perform an IMAGE level retrieve for all images belonging to patient Three^Secondary Capture Image.

12.4 Worklist SOP Class

Table 14-4 describes the example Worklist Management SOP Class DICOM(Super)Scripts.

TABLE: 14-4 Worklist Management SOP Class DICOM(Super)Scripts				
SCU Session		SCP Session		Scenario
My Documents\DVTK\Projects\Examples \scripts\worklist\scu Session File: test001.ses		My Documents\DVTK\Projects\Examples \scripts\worklist\scp Session File: test001.ses		
File Pane DICOM(Super)Script	Results File	File Pane DICOM(Super)Script	Results File	
1.ds	Summary_001_1_ ds_res.xml	1.ds	Summary_001_1_ ds_res.xml	Simulate request for worklist for <i>CT</i> modality with name <i>DVT</i> for today's date. Return Patient Names, Patient IDs and Study Instance UIDs. NOTE: Maximum-Length-Received for SCP is restricted to 4096 bytes.

12.5 Performed Procedure Step SOP Class

Table 14-5 describes the example Performed Procedure Step SOP Class DICOM(Super)Scripts.

TABLE: 14-5 Performed Procedure Step SOP Class DICOM(Super)Scripts				
SCU Session		SCP Session		Scenario
My Documents\DVTK\Projects\Examples \scripts\mpps\scu Session File: test001.ses		My Documents\DVTK\Projects\Examples \scripts\mpps\scp Session File: test001.ses		
File Pane DICOM(Super)Script	Results File	File Pane DICOM(Super)Script	Results File	
1.ds	Summary_001_1_ ds_res.xml	1.ds	Summary_001_1_ ds_res.xml	Simulate the N-CREATE of the MPPS with status "IN PROGRESS".
2.ds	Summary_001_2_ ds_res.xml	2.ds	Summary_001_2_ ds_res.xml	Simulate the N-SET of the MPPS with status "COMPLETED"

12.6 Storage Commitment SOP Class

Table 14-6 describes the example Storage Commitment SOP Class DICOM(Super)Scripts.

TABLE: 14-6 Storage Commitment SOP Class DICOM(Super)Scripts				
SCU Session		SCP Session		Scenario
My Documents\DVTK\Projects\Examples \scripts\commit\scu Session File: test001.ses		My Documents\DVTK\Projects\Examples \scripts\commit\scp Session File: test001.ses		
File Pane DICOM(Super)Script	Results File	File Pane DICOM(Super)Script	Results File	
1.ds	Summary_001_1_ ds_res.xml	1.ds	Summary_001_1_ ds_res.xml	Simulate a Storage Commitment transaction for 4 images.

12.7 Structured Reporting SOP Class

Table 14-7 describes the example Storage Commitment SOP Class DICOM(Super)Scripts.

TABLE: 14-7 Basic Structured Reporting SOP Class DICOM(Super)Scripts				
SCU Session		SCP Session		Scenario
My Documents\DVTK\Projects\Examples \scripts\report\scu Session File: test001.ses		My Documents\DVTK\Projects\Examples \scripts\report\scp Session File: test001.ses		
File Pane DICOM(Super)Script	Results File	File Pane DICOM(Super)Script	Results File	
1.ds	Summary_001_1_ ds_res.xml	1.ds	Summary_001_1_ ds_res.xml	Simulate creation and storage of a Basic Text Structured Report.

12.8 Print Management SOP Class

Table 14-8 describes the example Print Management SOP Class DICOM(Super)Scripts.

TABLE: 14-8 Print Management Meta SOP Class DICOM(Super)Scripts				
SCU Session		SCP Session		Scenario
My Documents\DVTK\Projects\Examples \scripts\print\scu Session File: test001.ses		My Documents\DVTK\Projects\Examples \scripts\print\scp Session File: test001.ses		
File Pane DICOM(Super)Script	Results File	File Pane DICOM(Super)Script	Results File	
1.ds	Summary_001_1_ ds_res.xml	1.ds	Summary_001_1_ ds_res.xml	Simulate SCP rejecting Associate Request due to “ <i>application context name not supported</i> ”.
2.ds	Summary_001_2_ ds_res.xml	2.ds	Summary_001_2_ ds_res.xml	Basic Grayscale Print session using a STANDARD\1,2 film format. Send 2 images.

12.9 Verification SOP Class

Table 14-9 describes the example Verification SOP Class DICOM(Super)Scripts.

TABLE: 14-9 Verification SOP Class DICOM(Super)Scripts				
SCU Session		SCP Session		Scenario
My Documents\DVTK\Projects\Examples \scripts\verifcn\scu Session File: test001.ses		My Documents\DVTK\Projects\Examples \scripts\verifcn\scp Session File: test001.ses		
File Pane DICOM(Super)Script	Results File	File Pane DICOM(Super)Script	Results File	
1.ds	Summary_001_1_ ds_res.xml	1.ds	Summary_001_1_ ds_res.xml	Simulate a simple C-ECHO-RQ / C-ECHO-RSP message exchange.

13. Appendices

13.1 DICOMScript Language Reference

The DICOMScript Language supports the following commands:

Commands with respect to DICOMObject / Attribute manipulation:

- ◆ COMPARE / COMPARE_NOT - Compare attributes.
- ◆ COPY - Copy attribute.
- ◆ CREATE - Create a DICOMObject in the data-warehouse.
- ◆ DELETE - Delete a DICOMObject from the data-warehouse.
- ◆ DISPLAY - Display attribute selection or DICOMObject from the data-warehouse.
- ◆ EXPORT - Export a DICOMObject from the data-warehouse over a network Association.
- ◆ IMPORT - Import a DICOMObject over a network Association into the data-warehouse.
- ◆ READ - Read a DICOMObject from a Media Storage file into the data-warehouse.
- ◆ RECEIVE - Receive and validate a DICOMObject over a network Association.
- ◆ RESET - Reset the Dataware House content and/or Association State.
- ◆ SEND - Send a DICOMObject over a network Association.
- ◆ SET - Set DICOMObject Attribute values.
- ◆ VALIDATE - Validate a DICOMObject in the data-warehouse.
- ◆ WRITE - Write a DICOMObject to a Media Storage file.

Script Execution Context commands:

- ◆ ADD-GROUP-LENGTH - Enable group lengths.
- ◆ APPLICATION-ENTITY - Define the Application Entity for definition file use.
- ◆ DEFINE-SQ-LENGTH - Enable defined sequence lengths.
- ◆ POPULATE - Adds Type 2 attributes during DICOMObject encoding.
- ◆ STRICT-VALIDATION - Enable strict validation.
- ◆ VALIDATION - Define validation type required.

Utility commands:

- ◆ CONFIRM - Request User confirmation for an action.
- ◆ DELAY - Delay further DICOMScript execution.
- ◆ ECHO - Display prompt / information to User.
- ◆ SYSTEM - Perform specified operating system call.
- ◆ TIME - Display the current system time.
- ◆ VERBOSE - Define verbosity during DICOMScript execution.

13.1.1 ADD-GROUP-LENGTH ON / OFF

Syntax

- ADD-GROUP-LENGTH ON
- ADD-GROUP-LENGTH OFF

13.1.2 APPLICATION-ENTITY

Syntax

- APPLICATION-ENTITY *<ae_name> <ae_version>*

13.1.3 COMPARE / COMPARE_NOT

Syntax

- COMPARE *<dicom_object_ref> <tag> <dicom_object_ref> <tag>*
- COMPARE_NOT *<dicom_object_ref> <tag> <dicom_object_ref> <tag>*

13.1.4 CONFIRM

Syntax

- CONFIRM

13.1.5 COPY

Syntax

- COPY *<destination dicom_object_ref> <tag> <source dicom_object_ref> <tag>*

13.1.6 CREATE

Syntax for creating ACSEObjects

- CREATE ASSOCIATE-RQ
- CREATE ASSOCIATE-AC
- CREATE ASSOCIATE-RJ
- CREATE RELEASE-RQ
- CREATE RELEASE-RP
- CREATE ABORT-RQ

Syntax for creating DICOM Command, Dataset and Item DICOMObjects

- CREATE *<command_ref>*
- CREATE *<command_ref> <dataset_ref>*
- CREATE *<item_ref> [DEFINED_LENGTH]*

Syntax for creating an Item reference

- CREATE *<sequence_address> <item_ref>*

This command creates a reference to an item in another DICOM object addressed by *<sequence_address>*.

In this case there is no object explicitly created – the CREATE command simply adds an extra *<item_ref>* which refers to the item addressed by *<sequence_address>*.

<sequence_address> (*<dataset_ref> <tag> [<item_nr>]*)
 | (*<sequence_address> <tag> [<item_nr>]*)
 - *<dataset_ref>* identifies the DICOMObject containing the item.
 - *<tag>* identifies the sequence tag in *<dataset_ref>*.

<item_nr> [*<integer>*]
 | # nothing - default to 1st item
 - identifies which item (0..n) in a sequence tag is addressed.

The *<sequence_address>* allows recursion to access items within nested sequences.

Syntax for creating Media Storage objects

- **CREATE FILE-HEAD**
- **CREATE FILE-TAIL**

13.1.7 DEFINE-SQ-LENGTH ON / OFF

Syntax

- **DEFINE-SQ-LENGTH ON**
- **DEFINE-SQ-LENGTH OFF**

13.1.8 DELAY

Syntax

- **DELAY <integer>**

13.1.9 DELETE

Syntax for deleting ACSEObjects

- **DELETE ASSOCIATE-RQ**
- **DELETE ASSOCIATE-AC**
- **DELETE ASSOCIATE-RJ**
- **DELETE RELEASE-RQ**
- **DELETE RELEASE-RP**
- **DELETE ABORT-RQ**

Syntax for deleting DICOM Command, Dataset and Item DICOMObjects

- **DELETE <command_ref>**
- **DELETE <dataset_ref>**
- **DELETE <item_ref>**

Syntax for deleting Media Storage objects

- **DELETE FILE-HEAD**
- **DELETE FILE-TAIL**

13.1.10 DISPLAY

Syntax

- **DISPLAY <dicom_object_ref> [<tag> [<tag>]...]**

13.1.11 ECHO

Syntax

- **ECHO <string 256>**

13.1.12 EXPORT

Syntax

- **EXPORT** *<dicom_message>*

<dicom_message> *<command_ref>*
 | *<command_ref>* *<dataset_ref>* [*<pres_context_id>*]

<pres_context_id> **PRESENTATION-CONTEXT-ITEMS**(*<integer>*)

13.1.13 IMPORT

Syntax

- **IMPORT** *<dicom_message>*

<dicom_message> *<command_ref>*
 | *<command_ref>* *<dataset_ref>*

13.1.14 POPULATE ON / OFF

Syntax

- **POPULATE ON**
- **POPULATE OFF**

13.1.15 READ

Syntax

- **READ** *filename* *<dataset_ref>*
- **READ** *filename* *<tag>*

13.1.16 RECEIVE

Syntax for receiving ACSE objects

- **RECEIVE ASSOCIATE-RQ** [*<assoc_rq_param_list>*]
- **RECEIVE ASSOCIATE-AC** [*<assoc_ac_param_list>*]
- **RECEIVE ASSOCIATE-RJ** [*<assoc_rj_param_list>*]
- **RECEIVE RELEASE-RQ**
- **RECEIVE RELEASE-RP**
- **RECEIVE ABORT-RQ** [*<abort_rq_param_list>*]

Syntax for receiving Command and Dataset DICOMObjects

RECEIVE *<dicom_message>*

<dicom_message> *<command>* [*<dicom_object_attribute_list>*]
 | *<command>* *<dataset_name>* [*<dicom_object_attribute_list>*]

13.1.17 RESET

Syntax

- **RESET ALL**

- RESET WAREHOUSE
- RESET RELATION
- RESET ASSOCIATION
- RESET SCRIPT-EXECUTION-CONTEXT

13.1.18 SEND

Syntax for sending ACSE objects

- SEND ASSOCIATE-RQ [<assoc_rq_param_list>]
- SEND ASSOCIATE-AC [<assoc_ac_param_list>]
- SEND ASSOCIATE-RJ [<assoc_rj_param_list>]
- SEND RELEASE-RQ
- SEND RELEASE-RP
- SEND ABORT-RQ [<abort_rq_param_list>]

Syntax sending Command and Dataset DICOM Objects

SEND <dicom_message>

<dicom_message> <command> [<dicom_object_attribute_list>]
| <command> <dataset_name> [<pres_context_id>] [<dicom_object_attribute_list>]

<pres_context_id> PRESENTATION-CONTEXT-ITEMS (<integer>)

13.1.19 SET

Syntax for setting ACSE Object Parameters

- SET ASSOCIATE-RQ <assoc_rq_param_list>
- SET ASSOCIATE-AC <assoc_ac_param_list>
- SET ASSOCIATE-RJ <assoc_rj_param_list>
- SET ABORT-RQ <abort_rq_param_list>

Syntax for setting Command, Dataset and Item DICOM Object Attributes

- SET <command_ref> <dicom_object_attribute_list>
- SET <dataset_ref> <dicom_object_attribute_list>
- SET <item_ref> <dicom_object_attribute_list>

Syntax for setting Media Storage objects

- SET FILE-HEAD <file_head_param_list>
- SET FILE-TAIL <file_tail_param_list>

13.1.20 STRICT-VALIDATION ON / OFF

Syntax

- STRICT-VALIDATION ON
- STRICT-VALIDATION OFF

13.1.21 SYSTEM

Syntax

- **SYSTEM** <*string 256*>

13.1.22 TIME

Syntax

- **TIME**

13.1.23 VALIDATE

Syntax

VALIDATE <*dicom_object*> <*reference_dicom_object_list*>

<*reference_dicom_object_list*> = <*dicom_object*> [OR <*dicom_object*> ...]

13.1.24 VALIDATION

Syntax

- **VALIDATION ENABLED**
- **VALIDATION ENABLED-USE-DEF-ONLY**
- **VALIDATION ENABLED-USE-VR-ONLY**
- **VALIDATION ENABLED-USE-REF-ONLY**
- **VALIDATION ENABLED-USE-DEF-AND-VR**
- **VALIDATION ENABLED-USE-DEF-AND-REF**
- **VALIDATION ENABLED-USE-VR-AND-REF**
- **VALIDATION DISABLED**

13.1.25 VERBOSE ON / OFF

Syntax

- **VERBOSE ON**
- **VERBOSE OFF**

13.1.26 WRITE

Syntax for writing Dataset DICOMObjects

- **WRITE** *filename* <*dataset_ref*>

Syntax for writing Media Storage objects

- **WRITE** *filename* **FILE-HEAD**
- **WRITE** *filename* **FILE-TAIL**

13.2 Definition File Format Reference

The Definition File format can formally be described in Backus-Naur Form (BNF).

With BNF notation the syntactic specification of a grammar can be described. A grammar involves four quantities: *terminals*, *nonterminals*, a *start symbol* and *productions*.

The basic symbols of which strings in the language are composed are called *terminals*. Terminals are upper-case letters, digits, operator or punctuation symbols such as '(', ')', or ','.

Non-terminals are special symbols that denote sets of strings (or syntactic categories). Nonterminals are lower-case letters enclosed in brackets.

One non-terminal is selected as the *start symbol* (<language>) and it denotes the language in which we are truly interested.

The *productions* define ways in which the syntactic categories may be built up from one another and from the terminals. Each production consists of a nonterminal, followed by a symbol '::=', followed by a sequence of non-terminals and terminals. The vertical bar symbol '|' means logical OR.

TOP LEVEL DEFINITIONS

```
<definitiongrammar> ::= <definitioncomponents>

<definitioncomponents> ::= <definition>
                           | <definitioncomponents> <definition>

<definition> ::= <define>
                 | <displayformat>

<define> ::= DEFINE <definitions> ENDDEFINE

<definitions> ::= <systemdef>
                  | <metasopclassdef>
                    | <commanddef>
                    | <commandioddef>
                    | <macrodef>
```

SYSTEM DEFINITION

```
<systemdef> ::= SYSTEM <system name> <system version> <ae definition>

<ae definition> ::= <ae name> <ae version>

<ae name> ::= <string>

<ae version> ::= <string>
```

METASOPCLASS DEFINITION

```
<metasopclassdef> ::= METASOPCLASS <beginmetasopclassdef> <sopclassdeflist>

<beginmetasopclassdef> ::= <metasopclassuid> <metasopclassname>

<metasopclassuid> ::= <string>

<metasopclassname> ::= <string>

<sopclassdeflist> ::= <sopclassdef>
                     | <sopclassdeflist> <sopclassdef>
```

COMMAND DEFINITION

```
<commanddef> ::= <dimsecmddef> <attributedeflist>

<dimsecmddef> ::= <dimsecmd>
```

<dimsecmd> ::= <command>

MACRO DEFINITION

<macrodef> ::= **MACRO** <macroname> <attributedeflist>

<macroname> ::= <string>

COMMAND IOD DEFINITION

<commandioddef> ::= <dimsecmddef> <iodname> <iodcontentdef>

<iodname> ::= <string>

<iodcontentdef> ::= <sopclassdef> <moduledeflist>

SOPCLASS DEFINITION

<sopclassdef> ::= **SOPCLASS** <sopclassuid> <sopclassname>

<sopclassuid> ::= <string>

<sopclassname> ::= <string>

MODULE DEFINITION

<moduledeflist> ::= <moduledef>
| <moduledeflist> <moduledef>

<moduledef> ::= <modulename> <attributedeflist>

<modulename> ::= **MODULE** <modulename> <moduleusage>

<modulename> ::= <string>

<moduleusage> ::= <usage> <condition>

<attributedeflist> ::= <attributedef>
| <attributedeflist> <attributedef>

ATTRIBUTE DEFINITION

<attributedef> ::= (<attributetagdef> , <attributetype> , <attributevaluedef>)
| <attributename> <condition>
| <macroreference>

<attributetagdef> ::= <attributetag>

<attributetag> ::= <hex>

<attributetype> ::= <integer>
| <type>

<attributename> ::= <string>

<attributevaluedef> ::= <sequencevaluedef>
| <othervaluedef>

<sequencevaluedef> ::= <sequenceintro> <itemattributedeflist>
| <sequenceintro>

<sequenceintro> ::= **SQ** , <sequencevm> ,

<sequencevm> ::= <attributevm>
| <attributevm> , <sequenceim>

<itemattributedeflist> ::= <itemattributedef>
| <itemattributedeflist> <itemattributedef>

<itemattributedef> ::= > <attributedef>

<attributevm> ::= <vm>
| <vm> : <vm>

<sequenceim> ::= <vm>
| <vm> : <vm>

<vm> ::= <integer>
| <integer> **n**
| **n**

<attributevr> ::= <vr>
| <vr> / <vr>

<othervaluedef> ::= <attributevr> , <attributevm> <valuesdeflist>

<valuesdef> ::= ε
| , <firstvaluesdef>
| , <firstvaluesdef> , <othervaluedeflist>

<firstvaluesdef> ::= <valuetype> , <valueslist>

<othervaluedeflist> ::= <othervaluedef>
| <othervaluedeflist> , <othervaluedef>

<othervaluedef> ::= <valuetype> , <valueslist>
| <valueslist>

<valueslist> ::= <value>
| <valueslist> | <value>

<value> ::= <string>
| <hex>
| <integer>

<valuetype> ::= **D** (defined term)
E (enumerated value)
DL (defined term list – defined terms apply to all attribute values)
EL (enumerated value list – enumerated values apply to all attribute values)

<macroreference> ::= **INCLUDEMACRO** <macroname> <macrocondition>

<macrocondition> ::= <condition>

CONDITION DEFINITION

<condition> ::= ε
| ' ' <string>
| ' ' <conditionexpression>
| ' ' **WEAK** (<conditionexpression> , <conditionexpression>)

<conditionexpression> ::= <compoundexpression>
| <conditionexpression> **OR** <compoundexpression>

<compoundexpression> ::= <expression>
| <expression> **AND** <compoundexpression>

<simpleexpression> ::= **PRESENT** <attributetagaddress>
| **EMPTY** <attributetagaddress>
| **VALUE** <attributetagaddress> <valuenumber> <operator> <conditionvalue>
| **NOT** <simpleexpression>
| (<conditionexpression>)
| **TRUE**
| **FALSE**

<valuenumber> ::= ε
| <integer>

<conditionvalue> ::= <string>

```

<operator> ::= =
            | <
            | >
            | <=
            | >=

```

```

<attributetagaddress> ::= <attributetag>
                        | <attributetagpath>

```

```

<attributetagpath> ::= <attributetaghierarchy> <attributetag>
                      | <attributetaguppath> <attributetag>
                      | <attributetagdownpath> <attributetag>
                      | <attributetaghierarchy> <attributetagdownpath> <attributetag>
                      | <attributetaguppath> <attributetagdownpath> <attributetag>

```

```

<attributetaghierarchy> ::= ./

```

```

<attributetaguppath> ::= <attributetagup>
                       | <attributetaguppath> <attributetagup>

```

```

<attributetagup> ::= ../

```

```

<attributetagdownpath> ::= <attributetagdown>
                          | <attributetagdownpath> <attributetagdown>

```

```

<attributetagdown> ::= / <attributetag>

```

IMAGE DISPLAY FORMAT DATA FILE

```

<displayformat> ::= <imagedisplayformat>
                  | <annotationdisplayformatid>

```

```

<imagedisplayformat> ::= IMAGE-DISPLAY-FORMAT <string> <integer>

```

```

<annotationdisplayformatid> ::= ANNOTATION-DISPLAY-FORMAT-ID <string> <integer>

```

13.3 ADVT Compatibility Notes

DVT was designed to be compatible with ADVT sessions, scripts and definitions, but there are differences that require attention.

13.3.1 ADVT Emulator Sessions

ADVT emulator sessions will not work directly in DVT. DVT has added a **SESSION-TYPE** attribute to the Session File. When DVT opens an ADVT session file, it assumes that it is a script session. The recommended solution to this problem is to recreate the session file in DVT. The ADVT session can be used by adding the line

SESSION-TYPE emulator

to the start of the session file.

13.3.2 ADVT Definitions

DVT has made many improvements to definition file structure and content, so it is recommended that only the DVT definition files be used.

13.3.3 Backslash Handling

There is an incompatibility between ADVT DICOMScripts and DVT DICOMScripts when using the BACKSLASH character in an attribute value. In ADVT the BACKSLASH was represented by a single "\" in the attribute value. In DVT the BACKSLASH must be represented by a double "\\" in the attribute value. If this change is not made then the single "\" is converted by the script parser to "\?". The exceptions to this are in filenames and in the Image Display Foramt, DICOM tag (2010,0010).

13.4 Third Party Copyright Notices

13.4.1 UC Davis DICOM Library

DVT is originally developed from a snapshot of the UC Davis DICOM Library. The following is the license for the library:

```

/*****
    Copyright (C) 1995, University of California, Davis

    THIS SOFTWARE IS MADE AVAILABLE, AS IS, AND THE UNIVERSITY
    OF CALIFORNIA DOES NOT MAKE ANY WARRANTY ABOUT THE SOFTWARE, ITS
    PERFORMANCE, ITS MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR
    USE, FREEDOM FROM ANY COMPUTER DISEASES OR ITS CONFORMITY TO ANY
    SPECIFICATION. THE ENTIRE RISK AS TO QUALITY AND PERFORMANCE OF
    THE SOFTWARE IS WITH THE USER.

    Copyright of the software and supporting documentation is
    owned by the University of California, and free access
    is hereby granted as a license to use this software, copy this
    software and prepare derivative works based upon this software.
    However, any distribution of this software source code or
    supporting documentation or derivative works (source code and
    supporting documentation) must include this copyright notice.
*****/

/*****
 *
 * University of California, Davis
 * UCDMC DICOM Network Transport Libraries
 * Version 0.1 Beta
 *
 * Technical Contact: mhoskin@ucdavis.edu
 *
 *****/

```

13.4.2 OpenSSL

DVT uses the OpenSSL library to provide secure sockets support. The following is the license for the library.

LICENSE ISSUES
=====

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License

```

/* =====
 * Copyright (c) 1998-2003 The OpenSSL Project. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:

```

```

*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. All advertising materials mentioning features or use of this
*    software must display the following acknowledgment:
*    "This product includes software developed by the OpenSSL Project
*    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
*
* 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
*    endorse or promote products derived from this software without
*    prior written permission. For written permission, please contact
*    openssl-core@openssl.org.
*
* 5. Products derived from this software may not be called "OpenSSL"
*    nor may "OpenSSL" appear in their names without prior written
*    permission of the OpenSSL Project.
*
* 6. Redistributions of any form whatsoever must retain the following
*    acknowledgment:
*    "This product includes software developed by the OpenSSL Project
*    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
*
* THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
* =====
*
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com). This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
*
*/

Original SSLeay License
-----

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to. The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code. The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.
* This can be in the form of a textual message at program startup or

```

```
* in documentation (online or textual) provided with the package.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the copyright
*   notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*   notice, this list of conditions and the following disclaimer in the
*   documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
*   must display the following acknowledgement:
*   "This product includes cryptographic software written by
*    Eric Young (eay@cryptsoft.com)"
*   The word 'cryptographic' can be left out if the routines from the library
*   being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof) from
*   the apps directory (application code) you must include an acknowledgement:
*   "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed. i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/
```