1512-6

# RESEARCHING XML AND RELATED TECHNOLOGIES

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of
Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering, Computer Engineering Program

151236

by

Deniz KILINÇ

June, 2004
İZMİR

# Ms.Sc. THESIS EXAMINATION RESULT FORM

We certify that we have read the thesis, entitled "Reseaching XML and Related Technologies" completed by Deniz KILINÇ under supervision of Prof.Dr. Recep Alp KUT and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Alp KUT

[illegible]

Supervisor

Doç. Dr. Yalçın ÇEBİ

[illegible]

Committee Member

Doç. Dr. Ender Y. Bulgun

[illegible]

Committee Member

Approved by the

Graduate School of Natural and Applied Sciences

[illegible]

Prof.Dr.Cahit Helvacı

Director

# ACKNOWLEDGMENTS

Firstly, I would like to thank my advisor Prof.Dr. Recep Alp KUT for his advises and great guidance. He has always motivated me. Secondly, I want to thank my wife Sibel KILINÇ and my family for their understanding. Because, during the writing of this thesis, I was not be able to interested in them.

Finally, thank to W3C (World Wide Web Consortium) XML working group for the development of most powerful integration and web technology XML.

Deniz KILINÇ

# ABSTRACT

This thesis is a research for XML and its related technologies. XML is easy to learn, implement, read, and test. It has shortened product development time for most XML-related and data exchange projects like B2B.

XML, XSL, XSLT, XPath and DTD technologies are examined deeply and explained with rich samples. In addition to these technologies, XML's EBNF architecture is with its 89 EBNF rules and real life XML application scenarios are studied. It is also cleared that XML technology is not a programming language or not a database alone.

**Keywords**: XML, DTD, XSL, XPath, XSLT, EBNF, SGML, HTML.

# ÖZET

Bu tez, XML ve ilgili teknolojilerini inceleyen bir çalışmadır. XML öğrenmesi, okunması, uygulanması ve test edilmesi kolay bir teknolojidir. B2B gibi uygun uygulamalarda geliştirme zamanını kısaltır.

XML, XSL, XSLT, XPath ve DTD teknolojileri detaylı olarak incelenmiş ve zengin örneklerle açıklamalar getirilmiştir. Bu teknolojilere ek olarak, XML teknolojisinin 89 tane EBNF kuralını içeren EBNF altyapısı ve gerçek yaşamdaki XML uygulamaları ortaya konulmuştur. Çeşitli avantajlarına ve XSL gibi yardımcı teknolojilerine rağmen, XML teknolojisinin tek başına bir programlama dili veya veritabanı olmadığı görülecektir.

**Anahtar Sözcükler**: XML, DTD, XSL, XPath, XSLT, EBNF, SGML, HTML.

# CONTENTS

Page

Contents .......................................................................................v
List of Tables.................................................................................xi
List of Figures..............................................................................xviii

Chapter One

INTRODUCTION TO XML TECHNOLOGY

Chapter Two

XML TECHNOLOGY BASICS

2.1 Simply HTML........................................................................5
2.2 What is XML? .......................................................................8
2.3 Simply XML ..........................................................................9
2.4 Differences Between XML and HTML..............................10
2.5 Design Goals of XML.........................................................10
2.6 XML Features and Approaches .........................................12
2.6.1 Xml Has Structural Text Format ..................................12
2.6.2 XML Is Readable.............................................................13
2.6.3 XML Processing Is Easy And Cheap ............................14
2.6.4 XML Technology Supports Existing Security Solutions ..........14
2.6.5 XML Is Not A Programming Language ........................15
2.7 XML Real Life Scenarios....................................................17
2.7.1 Using XML Documents As Data Sources .....................17

2.7.2 Use of XML Documents For Integration of Different Systems .......... 18
2.7.3 Use of XML Documents for Standardization of User Interfaces ........ 20
2.7.4 XML As The Structural Content Of ActiveX Components ................ 22
2.7.5 XML For Defining Web Resources .................................................. 23

**Chapter Three**

**FIRST XML DOCUMENT**

3.1 Introduction .......................................................................................... 24
3.2 Creation of XML Document ................................................................ 24
3.3 Processing Instruction (PI) .................................................................. 25
3.4 Elements .............................................................................................. 26
3.5 Saving an XML Document .................................................................. 28
3.6 Displaying XML Document by a Browser .......................................... 29
3.7 Displaying XML Document by Using Cascade Style Sheet (CSS) ............... 31
3.8 Creation of CSS .................................................................................. 31
3.9 Integrating CSS Style Sheet with XML Document. ............................ 32

**Chapter Four**

**RESEARCHING XML 1.0 EBNF RULES**

4.1 BNF Grammar ..................................................................................... 34
4.2 Basic BNF ........................................................................................... 36
4.3 Well-Formed Documents ..................................................................... 37
4.4 Documents ........................................................................................... 37
4.5 Characters ............................................................................................ 38
4.6 Common Syntactic Constructs ............................................................ 40
4.7 Character Data and Markup ................................................................ 44
4.8 Comments ............................................................................................ 45
4.9 Processing Instructions ....................................................................... 46
4.10 CDATA Sections ................................................................................ 47
4.11 Prolog and Document Type Definitions ............................................ 48
4.12 Standalone Document Declaration .................................................... 56
4.13 White-Spaces ...................................................................................... 57
4.14 Language Identification ..................................................................... 58
4.15 Elements ............................................................................................. 59
4.16 Element Type Declarations ................................................................ 63
4.17 Element Content ................................................................................ 63

4.18 Mixed Content ..... 65
4.19 Attribute-List Declarations ..... 65
4.20 Attribute Types ..... 66
4.21 Enumerated Attributes ..... 68
4.22 Attribute Defaults ..... 69
4.23 Attribute-Value Normalization ..... 70
4.24 Conditional Sections ..... 71
4.25 Character And Entity References ..... 72
4.26 Entity Declarations ..... 74
4.27 Text Declaration ..... 76
4.28 Encoding Declarations ..... 77
4.29 Notation Declarations ..... 78
4.30 Character Classes ..... 79

**Chapter Five**

**XML DOCUMENTS**

5.1 Introduction ..... 82
5.2 Character And Markup Data ..... 82
5.3 Elements And Tags ..... 83
5.4 Attributes ..... 85
5.5 Processing Instructions ..... 86
5.6 Comments ..... 89
5.7 CDATA Sections ..... 89
5.8 Entity References ..... 90
5.9 Well-Formed XML Documents ..... 92

**Chapter Six**

**DTD TECHNOLOGY**

6.1 Introduction ..... 94
6.2 Valid Documents ..... 94
6.3 Document Type Declaration ..... 95
6.3.1 Internal DTD ..... 96
6.3.2 External DTD ..... 99
6.3.3 General External DTD ..... 100
6.3.4 Special External DTD ..... 103
6.3.5 Using External And Internal DTDs Together ..... 104

6.4 Element Declarations ........ 106
6.4.1 Elements with Normal Content ........ 106
6.4.2 Elements Has Only Character Data ........ 108
6.4.3 Elements Has Mixed Content ........ 108
6.4.4 Elements Have Any Kind Of Content Type ........ 110
6.4.5 Empty Elements ........ 110
6.5 Element Declarations ........ 112
6.6 Order Identifiers ........ 112
6.7 Alternative Identifiers ........ 115
6.8 Using Order and Alternative Identifiers By Grouping ........ 116
6.9 Specifying Exact Number of Elements in DTDs ........ 118
6.10 Attribute List Declaration ........ 119
6.11 Attribute Default Declarations ........ 122
6.12 Attribute Types ........ 123
6.12.1 Character Data Type(CDATA) ........ 124
6.12.2 Named Token Data Type (NMTOKEN) ........ 125
6.12.3 Named Token List Data Type (NMTOKENS) ........ 126
6.12.4 ID Data Type ........ 126
6.12.5 Reference To ID Data Types (IDREF) ........ 127
6.12.6 Reference Lists to Unique Identifiers (IDREFS) ........ 130
6.12.7 Enumerated Attribute Type ........ 132
6.12.8 ENTITY Attribute Type ........ 132
6.12.9 ENTITIES Attribute Type ........ 133
6.13 Entity Declarations ........ 134
6.13.1 General Internal Parsed Entities ........ 135
6.13.2 Predefined General Entities ........ 138
6.13.3 General Internal Entity References Usage Situations ........ 138
6.13.4 General External Parsed Entities ........ 139
6.13.5 General External Entity References Usage Situations ........ 141
6.13.6 General External Unparsed Entities ........ 141
6.13.7 Notation Declarations ........ 142
6.13.8 External Unparsed Entity Definitions ........ 143
6.13.9 Internal Parsed Parameter Entities ........ 146
6.13.10 External Parsed Parameter Entities ........ 148
6.14 Conditional Sections ........ 154
6.15 Character Entities ........ 155
6.16 Language Identifiers ........ 158
6.17 Document Validation Constraints ........ 159

# Chapter Seven
# INTRODUCTION TO XSL TECHNOLOGY

7.1 Introduction ... 161
7.2 XSL (Extensible Stylesheet Language) ... 161
7.3 XSLT (Extensible Stylesheet Language Transformations) ... 162
7.4 First XSL Example ... 163
7.5 XSLT Transformation and XML Document Structures ... 166
7.6 XSL Stylesheet Structures ... 167
7.6.1 XSL Declaration ... 167
7.6.2 Declaration of Transformation Type ... 168
7.6.3 Determination of Transformation Format as HTML ... 169
7.6.4 Determination of Transformation Format as Text ... 172
7.6.5 XSL Templates ... 174
7.6.6 <xsl:value-of select> Element ... 177
7.6.7 Processing XML Elements Iteratively ... 178
7.6.8 Conditional Processing ... 181
7.6.9 <xsl:if> Element Usage ... 182
7.6.10 <xsl:choose> Element Usage ... 183
7.6.11 Adding PIs to Output Document - xsl:processing-instruction ... 186
7.6.12 Adding Comments to Output Document - xsl:comment ... 187
7.6.13 Adding Elements to Output Document - xsl:element ... 188
7.6.14 Adding Attributes to Output Document - xsl:attribute ... 189
7.6.15 Adding Attribute Sets to Output Document - xsl:attribute-set ... 191
7.6.16 Copying Document Nodes - xsl:copy ... 192
7.6.17 Using Script Language – xsl:script, xsl:eval ... 193
7.6.18 Merging XSL Pages ... 195
7.6.18.1 Using <xsl:include> Element ... 196
7.6.18.2 Using <xsl:import> Element ... 197
7.6.19 <xsl:call-template> Element Usage ... 197
7.6.20 Using Parameters ... 198
7.6.21 Using <xsl:param> Element To Work With Parameters ... 198
7.6.22 Passing Parameters To Templates ... 202
7.6.23 General Parameters ... 203
7.6.24 Using <xsl:variable> Element To Work With Variables ... 203
7.6.25 General Variables ... 204
7.6.26 Difference Between XSLT Parameters And Variables ... 205
7.6.27 Optimization Over Complex Documents - <xsl:key> ... 205
7.6.28 Ordering XML Elements - <xsl:sort> ... 207
7.6.29 Converting Values To String Data – string() ... 210
7.6.30 Working With White-Spaces ... 211
7.6.31 Normalizing White-Spaces ... 212

7.6.32 XSLT String Functions ........ 213
7.6.33 Joining String Values – concat() ........ 214
7.6.34 Controlling Containing String Values – contains() ........ 215
7.6.35 Finding Number of Characters in Strings – string-length() ........ 216
7.6.36 Comparing Starting Substrings - starts-with() ........ 217
7.6.37 Choosing Substrings - substring() ........ 218
7.6.38 Choosing Substrings Before Substrings – substring-before() ........ 219
7.6.39 Choosing Substrings After Substrings – substring-after() ........ 220
7.6.40 Replacing String Characters – translate() ........ 220
7.6.41 Converting Values To Numeric Data – number() ........ 221
7.6.42 XSLT Number Functions ........ 223
7.6.43 XSLT round() Function ........ 223
7.6.44 XSLT Ceiling() Function ........ 224
7.6.45 XSLT Floor() Function ........ 224
7.6.46 XSLT Sum() Function ........ 225
7.6.47 XSLT decimal-format Function ........ 226
7.6.48 XSLT format-number Function ........ 227
7.6.49 Combining More Than One XSLT Documents ........ 229

CONCLUSION ........ 234

References ........ 237

# LIST OF TABLES

**Page**

Table 2.1 KIBELENET Camera Categories HTML sample ........ 5
Table 2.2 Basic HTML Tags (WEB_7, 2000) ........ 7
Table 2.3 XML and HTML differences ........ 10
Table 2.4 Use of XML Documents as DataBase ........ 18
Table 2.5 XMLHTTP Technology ........ 19
Table 2.6 SOAP Message ........ 20
Table 2.7 XAML (WEB_27, 2004) ........ 21
Table 2.8 XML Data of An Excel Application ........ 22
Table 3.1 First XML Document ........ 24
Table 3.2 Processing Instruction Line ........ 26
Table 3.3 Elements ........ 26
Table 3.4 Element syntax ........ 27
Table 3.5 Root Element and Sub-Elements ........ 27
Table 3.6 Cameras.css Style Sheet ........ 31
Table 3.7 Cameras.css processing instruction ........ 33
Table 3.8 First XML Document ........ 33
Table 4.1 Basic BNF Symbols and Characters (Harold E.R, 1999) ........ 36
Table 4.2 Document Rule (WEB_12, 2004) ........ 37
Table 4.3 Well-formed XML document ........ 38
Table 4.4 Non-well formed XML document ........ 38
Table 4.5 Character Range Rule (WEB_12, 2004) ........ 39
Table 4.6 White Space Rule (WEB_12, 2004) ........ 40
Table 4.7 Names and Tokens Rule (WEB_12, 2004) ........ 40
Table 4.8 Names and Tokens (WEB_12, 2004) ........ 41
Table 4.9 Valid and non-valid XML names ........ 41
Table 4.10 Names and Tokens Rule (WEB_12, 2004) ........ 42
Table 4.11 Attribute in NMTOKEN and CDATA types ........ 42
Table 4.12 Valid and non-valid Nmtokens ........ 42
Table 4.13 Literals Rule (WEB_12, 2004) ........ 42
Table 4.14 Entity usage ........ 43
Table 4.15 Valid and non-valid entities ........ 43
Table 4.16 Literals Rule (2) (WEB_12, 2004) ........ 43

Table 4.17 Valid and non-valid attributes ........ 43
Table 4.18 Literals Rule (3) (WEB_12, 2004) ........ 44
Table 4.19 Valid and Non-Valid SystemLiterals ........ 44
Table 4.20 Literals Rule (4) (WEB_12, 2004) ........ 44
Table 4.21 Character Data Rule (WEB_12, 2004) ........ 45
Table 4.22 Comments Rule (WEB_12, 2004) ........ 45
Table 4.23 Valid and non-valid comments ........ 46
Table 4.24 Processing Instructions Rule (WEB_12, 2004) ........ 46
Table 4.25 Valid and Non-valid PIs ........ 46
Table 4.26 CDATA Sections (WEB_12, 2004) ........ 47
Table 4.27 CDATA Usage ........ 47
Table 4.28 Valid and non-valid CDATA Sections ........ 48
Table 4.29 Prolog Rule (WEB_12, 2004) ........ 48
Table 4.30 Valid and non-valid Prologs ........ 48
Table 4.31 Prolog Rule (2) (WEB_12, 2004) ........ 49
Table 4.32 Valid and non-valid XML declarations ........ 49
Table 4.33 Prolog Rule (3) (WEB_12, 2004) ........ 49
Table 4.34 Valid and non-valid XML versions ........ 50
Table 4.35 Prolog Rule (4) (WEB_12, 2004) ........ 50
Table 4.36 XML document that references external entities ........ 50
Table 4.37 Document Type Definition Rule (WEB_12, 2004) ........ 51
Table 4.38 XML Sample which has all declarations ........ 52
Table 4.39 WFC for internal DTD declarations ........ 53
Table 4.40 External Subset Rule (WEB_12, 2004) ........ 54
Table 4.41 URI references ........ 54
Table 4.42 Using PE references in external subsets ........ 54
Table 4.43 Using internal and external declaration together ........ 55
Table 4.44 Standalone Document Declaration Rule (WEB_12, 2004) ........ 56
Table 4.45 Valid and non-valid Standalone attributes ........ 56
Table 4.46 White-spaces ........ 57
Table 4.47 Usage xml:space attribute ........ 57
Table 4.48 Language Identification Rule (WEB_12, 2004) ........ 58
Table 4.49 Valid and non-valid ISO639Code codes (WEB_12, 2004). ........ 58
Table 4.50 Valid and non-valid IanaCode codes (WEB_12, 2004) ........ 59
Table 4.51 Valid and non-valid UserCode codes ........ 59
Table 4.52 xml:lang attribute ........ 59
Table 4.53 Element Rule (WEB_12, 2004) ........ 60
Table 4.54 Element Rule (2) (WEB_12, 2004) ........ 60
Table 4.55 Valid and non-valid start-elements ........ 61
Table 4.56 Valid and non-valid attributes ........ 61
Table 4.57 Element Rule (3) (WEB_12, 2004) ........ 61
Table 4.58 Valid and non-valid close-tags ........ 62
Table 4.59 Element Rule (4) (WEB_12, 2004) ........ 62
Table 4.60 Element Rule (5) (WEB_12, 2004) ........ 62
Table 4.61 Valid and non-valid empty elements ........ 62

Table 4.62 Element Type Declaration Rule (WEB_12, 2004) ........ 63
Table 3.63 Element type definitions ........ 63
Table 4.64 Element Content Models Rule (WEB_12, 2004) ........ 64
Table 4.65 Choice usage ........ 64
Table 4.66 Seq usage ........ 64
Table 4.67 Mixed Content Declaration Rule (WEB_12, 2004) ........ 65
Table 4.68 Valid and non-valid mixed contents ........ 65
Table 4.69 Attribute-List Declaration Rule (WEB_12, 2004) ........ 66
Table 4.70 Attribute-list declarations ........ 66
Table 4.71 Attribute declarations ........ 66
Table 4.72 Attribute Types Rule (WEB_12, 2004) ........ 67
Table 4.73 Enumerated Attributes (WEB_12, 2004) ........ 68
Table 4.74 Valid and non-valid notation definitions ........ 68
Table 4.75 Valid and non-valid enumeration types ........ 69
Table 4.76 Attribute Defaults Rule (WEB_12, 2004) ........ 69
Table 4.77 Attribute-list declarations ........ 70
Table 4.78 Conditional Sections Rule (WEB_12, 2004) ........ 71
Table 4.79 INCLUDE sections ........ 71
Table 4.80 IGNORE sections ........ 72
Table 4.81 Conditional sections ........ 72
Table 4.82 Character Reference Rule (WEB_12, 2004) ........ 72
Table 4.83 Entity Reference Rule (WEB_12, 2004) ........ 73
Table 4.84 Valid and non-valid entity references ........ 74
Table 4.85 Valid and non-valid parameter entity references ........ 74
Table 4.86 Entity Declarations Rule (WEB_12, 2004) ........ 74
Table 4.87 Valid and non-valid general entity declarations (WEB_12, 2004) ........ 75
Table 4.88 Valid and non-valid parameter entity declarations ........ 75
Table 4.89 External Entities Rule (WEB_12, 2004) ........ 75
Table 4.90 Valid and non-valid external entity declarations ........ 76
Table 4.91 Text Declaration Rule (WEB_12, 2004) ........ 76
Table 4.92 Valid and non-valid text declarations ........ 77
Table 4.93 Well-formed External Parsed Entities Rule (WEB_12, 2004) ........ 77
Table 4.94 Encoding Declarations Rule (WEB_12, 2004) ........ 77
Table 4.95 Valid and non-valid encoding declarations ........ 78
Table 4.96 Notation Declarations Rule (WEB_12, 2004) ........ 78
Table 4.97 Valid and non-valid notation declarations ........ 78
Table 4.98 Characters Rule (WEB_12, 2004) ........ 79
Table 4.99 Characters Rule (2) (WEB_12, 2004) ........ 81
Table 5.1 Using empty elements ........ 84
Table 5.2 Valid and non-valid XML type names ........ 85
Table 5.3 Valid and non-valid attributes ........ 86
Table 5.4 Supported encodings ........ 87
Table 5.5 Usage of PIs ........ 88
Table 5.6 Valid and non-valid PIs ........ 88
Table 5.7 Usage of comments ........ 89

Table 5.8 Valid and non-valid comments ........ 89
Table 5.9 CDATA usage ........ 90
Table 5.10 Valid and non-valid CDATA sections ........ 90
Table 5.11 Predefined entities (WEB_12, 2004) ........ 91
Table 5.12 Entity reference declaration and usage ........ 91
Table 5.13 Root element ........ 92
Table 5.13 Starting and closing element ........ 92
Table 5.14 Nesting ........ 92
Table 5.15 Parameter entity references ........ 93
Table 6.1 DTD units (Pitts N., 2004) ........ 96
Table 6.2 Simple DTD architecture ........ 97
Table 6.3 Simple DTD usage ........ 97
Table 6.4 DTD comparing ........ 98
Table 6.5 Element Orders ........ 98
Table 6.6 Internal DTD Usage ........ 99
Table 6.7 XML declaration for external DTD ........ 100
Table 6.8 XML 1.0 version DTD declaration (WEB_38, 2004) ........ 100
Table 6.9 ISO639 codes ........ 101
Table 6.10 External general DTD usage ........ 102
Table 6.11 Special external DTD ........ 103
Table 6.12 External special DTD usage ........ 103
Table 6.13 External and internal DTD usage ........ 104
Table 6.14 Entities in external and internal DTDs ........ 105
Table 6.15 Elements with normal content ........ 107
Table 6.16 Valid and non-valid elements with normal content ........ 107
Table 6.17 Elements have sub-elements with normal content ........ 107
Table 6.18 Elements have only character data ........ 108
Table 6.19 Elements have mixed content ........ 108
Table 6.20 Valid and non-valid elements with mixed content ........ 109
Table 6.21 Elements have any kind of content type ........ 110
Table 6.22 Valid and non-valid elements with any kind of content. ........ 110
Table 6.23 Empty elements ........ 111
Table 6.24 Valid empty elements ........ 111
Table 6.25 XML 1.0 first EBNF rule ........ 112
Table 6.26 Element order and number identifiers (Harold E.R, 1999) ........ 112
Table 6.27 Element declaration using order identifiers ........ 113
Table 6.28 XML element usage ........ 113
Table 6.29 XML element usage with order and number identifiers ........ 114
Table 6.30 Order and number identifiers ........ 114
Table 6.31 Sub element usage that is not defined ........ 115
Table 6.32 Element declaration using alternative identifier ........ 115
Table 6.33 Element declaration using alternative identifier ........ 116
Table 6.34 Defining elements using order and alternative identifiers by grouping ... 116
Table 6.35 Using order and alternative identifiers by grouping ........ 116
Table 6.36 Using order and alternative identifiers by grouping ........ 117

Table 6.37 Using order and alternative identifiers by grouping (2) .......................... 118
Table 6.38 Expanding number constraint. .......................... 119
Table 6.39 Declaration of an attribute .......................... 120
Table 6.40 Using elements and attribute declarations together .......................... 120
Table 6.41 Defining more than one attribute .......................... 121
Table 6.42 Attribute default declarations (WEB_12, 2004) .......................... 122
Table 6.43 Attribute types and categories (WEB_12, 2004) .......................... 123
Table 6.44 Valid character data type declaration .......................... 124
Table 6.45 Non-valid character data types .......................... 124
Table 6.46 Valid and non-valid named tokens .......................... 125
Table 6.47 Valid named token declarations .......................... 125
Table 6.48 Valid named token list declaration and usage .......................... 126
Table 6.49 Valid and non-valid ID attribute types .......................... 126
Table 6.50 Non-valid ID attribute type declarations .......................... 127
Table 6.51 Valid ID attribute type declaration and usage .......................... 127
Table 6.52 Stock Module XML Design ID and IDREF usage .......................... 128
Table 6.53 IDREFS attribute type and usage .......................... 131
Table 6.54 ENUMERATION attribute type declaration and usage .......................... 132
Table 6.55 ENTITY attribute type declaration .......................... 133
Table 6.56 ENTITY attribute type usage .......................... 133
Table 6.57 ENTITIES attribute type and usage .......................... 133
Table 6.58 Types of entities (WEB_12, 2004) .......................... 135
Table 6.59 Internal parsed entity declaration .......................... 136
Table 6.60 Internal parsed entity definition and usage .......................... 136
Table 6.61 Using internal and external declarations together .......................... 137
Table 6.62 Predefined entities (WEB_12, 2004) .......................... 138
Table 6.63 Using entity references in entity declarations .......................... 138
Table 6.64 Using markup as the content of entities .......................... 139
Table 6.65 Cross dependent entity declarations .......................... 139
Table 6.66 Entity usage in element definition .......................... 139
Table 6.67 General external parsed entity definition .......................... 139
Table 6.68 External key file .......................... 140
Table 6.69 General external parsed entity usage .......................... 140
Table 6.70 ENTITY type usage .......................... 142
Table 6.71 External Unparsed Entity Declarations .......................... 144
Table 6.72 Referencing to external unparsed entities .......................... 145
Table 6.73 Internal parsed parameter entity definition .......................... 146
Table 6.74 Dependent parameter entity definitions .......................... 146
Table 6.75 Internal parsed parameter entity definitions .......................... 147
Table 6.76 Parameter entity usage in element definition .......................... 147
Table 6.77 Valid and non-valid parameter entity usages .......................... 148
Table 6.78 External parameter entity references StockMaster .......................... 149
Table 6.79 External parameter entity references StockOperation .......................... 150
Table 6.80 External parameter entity references StockTransaction .......................... 150
Table 6.81 External parameter entity references StockModule .......................... 152

Table 6.82 Using conditional sections ........ 154
Table 6.83 Using parameter entities or conditional sections ........ 155
Table 6.84 Using character entities ........ 156
Table 6.85 Language identifiers ........ 158
Table 7.1 First XSL and XSLT transformation ........ 163
Table 7.2 XSL declaration ........ 168
Table 7.3 Transformation formats and attributes (WEB_16, 2004) ........ 168
Table 7.4 Usage of HTML transformation format ........ 169
Table 7.5 XMLContent2.xsl document ........ 170
Table 7.6 Determination of Transformation Format as Text ........ 172
Table 7.7 Determination of Transformation Format as Text - 2 ........ 173
Table 7.8 Simple template usage ........ 174
Table 7.9 Traversing elements iteratively with xsl:apply-templates element ........ 175
Table 7.10 xsl:value-of element ........ 177
Table 7.11 xsl:for-each element ........ 178
Table 7.12 xsl:if conditional processing element ........ 182
Table 7.13 xsl:choose conditional processing element skeleton ........ 184
Table 7.14 xsl:choose element ........ 184
Table 7.15 xsl:processing-instruction element ........ 187
Table 7.16 xsl:comment element. ........ 188
Table 7.17 xsl:element usage ........ 188
Table 7.18 xsl:attribute usage ........ 190
Table 7.19 Direct attribute adding method ........ 190
Table 7.20 xsl:attribute-set usage ........ 191
Table 7.21 Identity transformation using xsl-copy ........ 192
Table 7.22 Script language usage ........ 193
Table 7.23 Using <xsl:include> element ........ 196
Table 7.24 Using <xsl:import> element ........ 197
Table 7.25 Using <xsl:call-template> element ........ 198
Table 7.26 Using <xsl:param> element ........ 199
Table 7.27 Using <xsl:param> element in attribute values ........ 200
Table 7.28 Using <xsl:param> element's select attribute ........ 200
Table 7.29 Using parameters ........ 201
Table 7.30 Passing parameters to templates ........ 202
Table 7.31 General parameter description ........ 203
Table 7.32 Using <xsl:variable> element ........ 204
Table 7.33 General variables ........ 205
Table 7.34 Using <xsl:key> element ........ 206
Table 7.35 Ordering elements ........ 208
Table 7.36 Result according to first ordering key ........ 209
Table 7.37 Result according to first and second ordering key ........ 210
Table 7.38 Using string() function ........ 210
Table 7.39 Working with white-spaces ........ 212
Table 7.40 Working with white-spaces ........ 213
Table 7.41 XSLT string functions (WEB_16, 2004) ........ 214

Table 7.42 Usage of concat() function....214
Table 7.43 Usage of contains() function....215
Table 7.44 Usage of string-length() function....217
Table 7.45 Usage of starts-with() function....217
Table 7.46 Usage of substring() function....218
Table 7.47 Usage of substring-before() function....219
Table 7.48 Usage of substring-after() function....220
Table 7.49 Usage of translate() function....221
Table 7.50 Using number() function....222
Table 7.51 XSLT number functions (WEB_16, 2004)....223
Table 7.52 Using round() function....223
Table 7.53 Using ceiling() function....224
Table 7.54 Using floor() function....224
Table 7.55 Using sum() function....225
Table 7.56 Attributes of <decimal-format> element (WEB_16, 2004)....226
Table 7.57 Pattern characters of format-number function (WEB_16, 2004)....228
Table 7.58 Usage of format-number function....228
Table 7.59 Usage of document function....229
Table 7.60 Usage of document function without parameters....233

# LIST OF FIGURES

**Page**

Figure 1.1 General view for markup languages.......... 3
Figure 2.1 KIBELENET HTML page presentation.......... 7
Figure 3.1 Camera categories tree hierarchy.......... 28
Figure 3.2 Saving XML documents.......... 29
Figure 3.3 Viewing XML documents via Internet Explorer.......... 30
Figure 3.4 Expanding and collapsing XML documents.......... 30
Figure 3.5 Display of XML Document by using CSS style sheet.......... 33
Figure 6.1 Hierarchical structure of entity types in DTD files.......... 135
Figure 6.2 Rendering of table 5.60 using Internet Explorer.......... 137
Figure 6.3 Web rendering of external parsed entities.......... 141
Figure 6.4 Rendering unparsed entities using browsers.......... 146
Figure 6.5 ASCII table's first 128 characters.......... 156
Figure 6.6 Web rendering of XML documents which have character entities.......... 157
Figure 7.1 XSLT Transformations.......... 163
Figure 7.2 First XSL without reference.......... 165
Figure 7.3 Simple transformation output.......... 165
Figure 7.4 XSLT processing steps.......... 180
Figure 7.5 Merging XSLT Documents.......... 232

# CHAPTER ONE
# INTRODUCTION TO XML TECHNOLOGY

In 1967, William Tunnicliffe gave a conference at Canada Government Press Office (WEB_1, 2002). It was about the necessity of separating content and format information which were mixed together in one document.

In 1968, a book publisher from New York, Stanley Rice, prepared a standard tag set, to standardize the book chapter formats (WEB_1, 2002). GCA (Graphic Communications Association) helped this project and standard tag set was developed. GCA GenCode (WEB_2, 2003) community joined the tagging and marking up logic.

In 1969, three people from IBM research team, Goldfarb, Mosher and Lorie were designed GML (Generalized Markup Language) which was accepted as the first formal markup language (WEB_1, 2002). The objective of GML is to be able to transport and share the legal documents.

In 1978, GML, ancestor of all markup languages, started to improve by special team under ANSI (American National Standard Institute). In 1986, GML markup language was named as SGML (Standard Generalized Markup Language) (WEB_3, 2002) and appointed as an international standard (ISO8879) by ISO (International Organization for Standardization).

SGML is a *meta language*, by meaning it has the capability of describing other languages (WEB_3, 2002). Like in all other meta languages, it defines some basic rules and validity constraints and lets you to create your own tag set using these rules. SGML has been used in complex industries such as flying and automotive. So SGML has been too complex for application development.

In 1989, HTML (Hypertext Markup Language) was designed as an *SGML Application* by Tim Berners-Lee and Anders Berlung (WEB_4, 2004). The main objective of the project was making easy sharing and transferring procedures for documents over *internet*. SGML application had the meaning of all HTML standard tags were designed using SGML markup language (WEB_3, 2002).

HTML language has been only used for data representation by providing standard tags for content, like header, font, image and table (WEB_6, 1999). Although there has been no possibility to define your own markup tags in HTML, learning and using it, in applications has been really easy, So, it has constituted the basis of new constructed *Web Architecture*.

Because of some HTML weakness like, not to be extensible, not to be a meta language, not to be designed for content management, necessity of a new markup language over internet. This new markup language has to been neither complex like SGML nor simple like HTML.

In 1996, W3C (World Wide Web Consortium) (WEB_5, 2004) was gathered to overcome this necessity and design new markup language XML. First design works of XML, started at the end of August and after intensive works, in a short time, in 11 weeks, XML's first draft version was announced at SGML'96 conference (WEB_1, 2002). Other formal details of XML taken one more year and in 1998 XML 1.0 was published as a standard markup language by W3C.

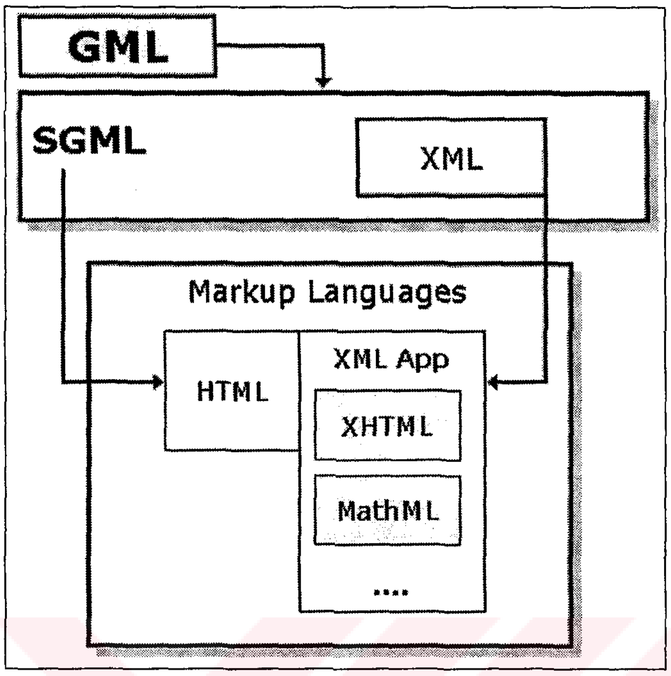


**Figure 1.1 General view for markup languages**

In figure 1.1, a general and historical view of markup languages is showed. After development of XML technology, other subset markup technologies like XHTML, MathML (WEB_8, 2004) and CML (WEB_9, 2004) have been also developed.

Thesis chapters and their contents are summarized below;

In the second chapter, basics of XML technology, XML and HTML differences, XML design goals, XML features-approaches and real life XML usage scenarios are explained.

In the third chapter, a simple XML document is created, examined with its document components, saved, displayed by the browser and formatted using a CSS style sheet on a browser.

In the fourth chapter, BNF grammar and its components are defined. XML 1.0 technology's 89 EBNF rules are declared and sampled.

In the fifth chapter, XML document's physical and logical structure units (elements, PI, comments, CDATA sections, and entity references) are tried to explain shortly.

In the sixth chapter, DTD (Document type definition) mechanism that specifies XML document's logical structure and controls document's markup units for validation purposes is told with much samples and explanations.

In the seventh chapter, XSL technology that is used to format and transform XML documents to another type of documents like HTML, WML and PDF is explained with real life samples.

Finally, in the last chapter, a conclusion about thesis is given.

# CHAPTER TWO
# XML TECHNOLOGY BASICS

## 2.1 Simply HTML

HTML (Hypertext Markup Language) is an SGML application which formats web pages (WEB_6, 1999). It has predefined standard tag set. Each tag has a special meaning for page formatting. Tags are rendered by web browsers and formatted content is presented. To be able to view web pages clearly in all browsers, tags or functions that are not standard should not be used within web pages.

When HTML pages firstly appeared, they were written in simple text editors. With the introduction of powerful web editors and code generators like Macromedia, all developers supposed themselves as web engineers. Main HTML page of a camera selling web site is showed below.

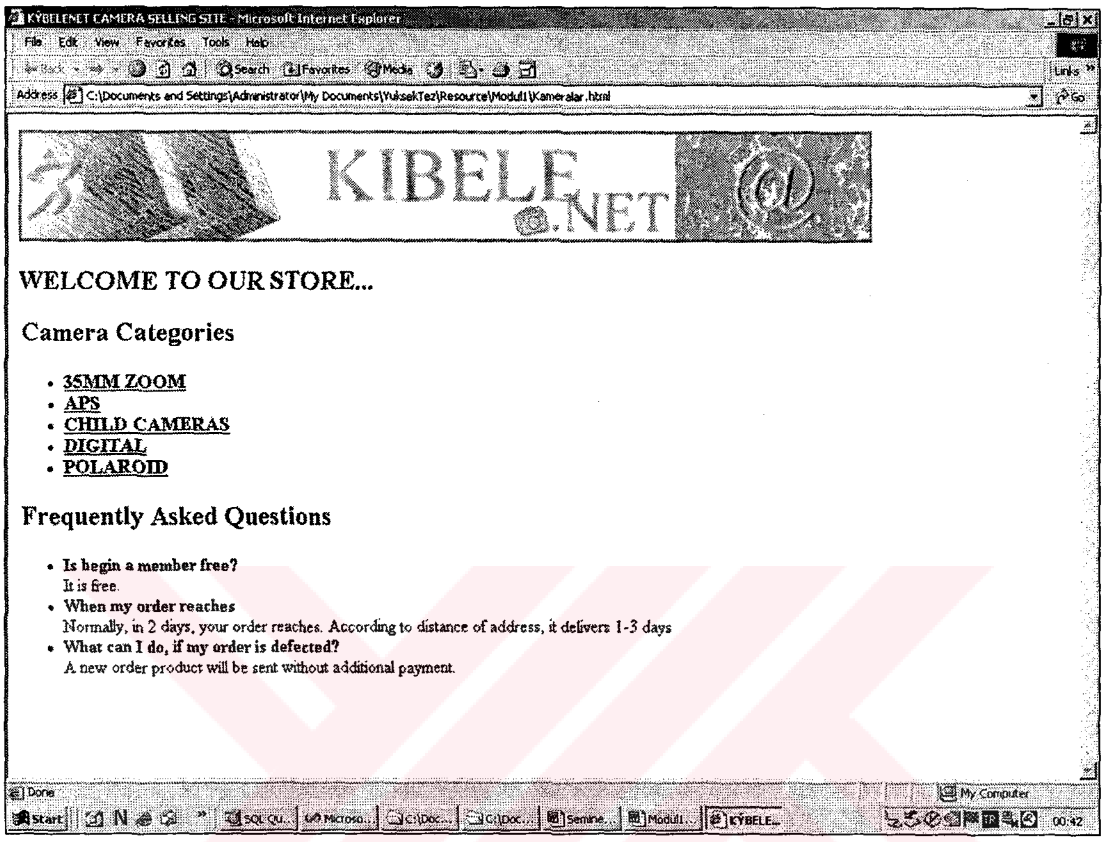
**Table 2.1 KIBELENET Camera Categories HTML sample**

```
<HTML>
     <HEAD>
       <TITLE>KİBELENET CAMERA SELLING SITE</TITLE>
     </HEAD>
     <BODY BGCOLOR="WHITE">
       <IMG SRC="BANNER.JPG"/>
       <P><H2>WELCOME TO OUR STORE...</H2></P>
       <TABLE BORDER=0>
         <TR><TD><H2>Camera Categories</H2></TD></TR>
```

**Table 2.1 Continued...**

```
        <TR>
          <TD><H3>
           <UL>
              <LI><A HREF="35MM.HTM"><B>35MM ZOOM</B></A></LI>
           <LI><A HREF="APS.HTM"><B>APS</B></A></LI>
           <LI><A HREF="CHILD.HTM">
                <B>CHILD CAMERAS</B></A></LI>
           <LI><A HREF="DIGITAL.HTM"><B>DIGITAL</B></A></LI>
           <LI><A HREF="POLAROID.HTM"><B>POLAROID</B></A></LI>
            </UL>
          </H3></TD>
        </TR>
        <TR><TD><H2>Frequently Asked Questions</H2></TD></TR>
        <TR>
          <TD>
            <UL>
              <LI><B>Is begin a member free?</B></LI>
              <BR/>It is free.
              <LI><B>When my order reaches</B></LI>
              <BR/> Normally, in 2 days, your order reaches.
                   According to distance of address, it delivers 1-3 days
              <LI><B>What can I do, if my order is defected? </B></LI>
              <BR/>
                A new order product will be sent without additional payment.
            </UL>
          </TD>
        </TR>
       </TABLE>
      </BODY>
</HTML>
```

Rendered view of the web page in Internet Explorer 6.0 is below.


KIBELENET CAMERA SELLING SITE - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Documents and Settings\Administrator\My Documents\YuksekTez\Resource\Modul1\Kameralar.html
Go Links
KIBELE@.NET
Done
My Computer
Start SQL Qu... Microso... C:\Doc... C:\Doc... Semine... Modul1... KIBELE...
00:42

WELCOME TO OUR STORE...

Camera Categories

- 35MM ZOOM
- APS
- CHILD CAMERAS
- DIGITAL
- POLAROID

Frequently Asked Questions

- **Is begin a member free?**
  It is free.
- **When my order reaches**
  Normally, in 2 days, your order reaches. According to distance of address, it delivers 1-3 days
- **What can I do, if my order is defected?**
  A new order product will be sent without additional payment.

**Figure 2.1 KIBELENET HTML page presentation**

Example HTML page contains different start-end tags and their attributes. Tags start with < character and end with > character. End tags also contain / character before the name of the element. Between start and end tag, element content is located. HTML tags are not interested with content, they only make representations.

**Table 2.2 Basic HTML Tags (WEB_7, 2000)**

| Tag | Usage |
|---|---|
| <HTML></HTML> | Specifies page as HTML page |
| <HEAD></HEAD> | Keeps header information |
| <BODY></BODY> | Keeps body information |
| <TITLE></TITLE> | Title information on the web browser |
| <H1><H2>...<H6> | Specifies text size |
| <P></P> | Specifies a paragraph |

| Table 2.2 Continued... | |
|---|---|
| <B></B> | Bolds the text |
| <IMG/> | Provides to show images |
| <A></A> | Provides to link other web pages |
| <TABLE></TABLE> | Provides to add tables |
| <TR></TR> | Adds rows to tables |
| <TD></TD> | Adds columns to tables |
| <UL></UL> | Specifies an unordered list |
| <LI></LI> | Specifies list items |
| <BR/> | Pages breaks |

## 2.2 What is XML?

XML (eXtensible Markup Language) (WEB_10, 2004) is a new simplified extensible markup language which has SGML's power and flexibility. XML is showed as a subset of SGML and uses markup tags, like HTML. Difference between XML and HTML is XML's power to define content's metadata. Like SGML, XML is also a meta language. But, although HTML becomes an SGML application, HTML is not a meta language. Some XML descriptions are below;

- *LISP* without functionality,
- *PDF* without acrobat,
- *EDI* without commercial semantic,
- *RTF* without word processing,
- *ZIP* without compression,
- *FLASH* without multimedia,
- *Morse Alphabet* with more characters,
- *Unicode* with more control characters,
- Microsoft's secret power against to Sun,
- Sun's open power against to Microsoft.

## 2.3 Simply XML

At first glance, XML technology which is developed by W3C (World Wide Web Consortium) (WEB_5, 2004) organization may be looked like the most popular web technology, HTML. Both technologies include some tags and properties. When details scraped, it is understood that the similarity between HTML and XML is only the name tag. HTML technology just formats the content but XML has special futures.

- Suitable with SGML.
- It holds both data and metadata which is data about data.
- Learning, reading, implementing and testing the XML technology is really easy.
- It shortens the application development time like B2B.
- Although XML is a simple document format, it is used in complex application scenarios like RPC messaging.
- XML is an open standard and developed from day to day. Also New auditing tools have been written.
- With the assistance of XSL, XSLT, XPath and XLink technologies, filtering, ordering, calculation and integration processes must be fulfilled
- Since XML technology supports internet and network technology, HTTP and SSL protocols are spontaneously supported.
- XML processing and traversing tools are world-wide and very cheap. With the development of DOM which is a tree based processing tool, document traversing have become a world standard.
- There is no need to prerequisite to read and understand XML documents.
- There is no need for high cost XML auditing editors.

## 2.4 Differences Between XML and HTML

**Table 2.3 XML and HTML differences**

| XML | HTML |
|---|---|
| Code Sample: <Personnel> <No>0001</No> <Name>Sibel KILINÇ</Name> <Telephone>053290194</Telephone> <Email>sibel@hotmail.com</Email> </Personnel> | Code Sample: <P> Personnel Information<BR/> <B>No: </B>0001 <B>Name:</B>Sibel KILINÇ <B>Telephone:</B>053290194 <B>Email:</B> sibel@hotmail.com </P> |
| Has an extensible structure. You can define your own tags. It has predefined rules. | Is not extensible. It has predefined element tags. So you can not define additional tags. |
| It keeps data and data about data (metadata) in structure manner. | Aim is to make presentation. There is no metadata. |
| Separates content from presentation. Presentation is done using XSL technology. | Content and presentation is mixed. |
| Readability is high. | There is no readability. |
| XML is a meta language. It is used to define other markup languages. | It is not a meta language. |
| Provides integration between different systems like in B2B. | It has no ability for integration. Because there is no semantic definition. |
| Parsing of XML documents are processed using DOM and SAX world standards. | Parsing is not important for HTML. |

## 2.5 Design Goals of XML

- "XML should be straightforwardly usable over the Internet" (WEB_11, 2004). This requirement seems almost to go without saying; of course you'd want a new markup language to be usable on the Internet, particularly the World Wide Web. But straightforwardly adds a subtle extra layer of meaning: like HTML, XML is not rocket science.

- "XML shall support a wide variety of applications" (WEB_11, 2004).
  XML technology does not only stores and represents data it can also be used for communication between different systems.

- "XML shall be compatible with SGML" (WEB_11, 2004).
  XML is not an SGML application; it is the subset of SGML. While the official definition of SGML is 400 pages, XML is only 26 pages. This small subset is compatible with SGML. Most of the SGML auditing tools can also be worked with XML documents.

- "It shall be easy to write programs which process XML documents" (WEB_11, 2004).
  Since SGML is a complex language, application development using SGML is very hard and time consuming. One important reason why XML programs need to be simple to write lies in the very extensibility of the language.

- "The number of optional elements in XML is to be kept to the absolute minimum, ideally zero" (WEB_11, 2004).
  While XML is a direct descendant of SGML (much more so than HTML), it does away with hundreds of optional "features" added to its parent over the course of many years.

- "XML documents should be human-legible and reasonably clear" (WEB_11, 2004).
  XML documents holds data and metadata in a structure manner. So, it is very easy to read and process XML documents.

- "The XML design should be prepared quickly" (WEB_11, 2004).
  Design duration of XML is not long as SGML which took 9 years. The basic design of XML was accomplished in eleven weeks. The work started in the last

few days of August 1996, and ended with the release of the first XML draft at the SGML '96 conference in November (WEB_1, 2002). So, XML's draft workings took 11 weeks.

- "The design of XML shall be formal and concise". (WEB_11, 2004).
  XML technology is constructed using EBNF (Extended Backus-Naur Form) grammar (WEB_10, 2004). BNF grammars are an outgrowth of compiler theory. A BNF grammar defines what is and is not a syntactically correct program or, in the case of XML, a syntactically correct document.

- "XML documents shall be easy to create". (WEB_11, 2004).
  There must be no prerequisites for XML development. It must be easy to create and manage XML documents.

- "Terseness in XML markup is of minimal importance". (WEB_11, 2004).
  The cost of this terseness is often clarity, determining what is intended at a given point where the optional markup might be expected. And the drafters of the XML spec set clarity as one of their guiding principles hence, this design goal.

## 2.6 XML Features and Approaches

### 2.6.1 Xml Has Structural Text Format

*Advantages:*

With this native feature, XML documents keep metadata, which says what data mean, in addition to content information. For example, information about a book in turn contains information about the title, author, chapters, b ody and index. Body c ontains paragraphs, line text, footnotes, etc. A document that describes a book has information that a person or machine can understand it. (WEB_13, 2004)

Most text files simply cannot offer this clear advantage. They either represent simply the content without metadata, or include metadata in a flat, one-level manner like .INI files. Files without metadata, must have separators like comma for processing. Since these files are not structured, readability is lost.

*Disadvantages:*

Although XML documents have a great advantage by keeping information and metadata together in a structured manner, using simple text files may be more efficient in some applications. (WEB_13, 2004)

### 2.6.2 XML Is Readable

*Advantages:*

That is one of the most talked advantages of XML. XML documents are readable both for human and machine. Readability not only makes debugging and diagnosis easier, but actually speeds up implementation time (WEB_13, 2004). Readability also provides writebility advantage by itself. There is no need to high-cost editors to write XML documents. You simply write them using “Microsoft Notepad” like text editors.

*Disadvantages:*

Main profit of XML technology is that is universal data exchange format between systems (WEB_5, 2004). Since humans may never need to actually touch or look at the XML document, after the application infrastructure has started to work once. But, for readability advantage, file size and network traffic utilization is increased.

If humans want, they can still create quite unreadable XML documents. Many developers code tags that having meaning for their application but are quite illegible to other humans (WEB_13, 2004). For example, element "<denopas981>" may not be meaningful to others.

### 2.6.3 XML Processing Is Easy And Cheap

*Advantages:*

One of the primary issue faced by alternative data file formats and database languages is that processing tools are expensive (WEB_13, 2004). XML technology has not had this disadvantage, because processing tools have become relatively widespread and inexpensive. Since XML is a structured document that shares many of the processing and parsing requirements, lots of parsers have been built. Many of these parsers are now built-in to general browsers. Document Object Model (DOM) (WEB_14, 2004) has been created by the W3C as a general model for how parsers and processors should interact and process XML documents for representation as trees. As a result, the DOM has produced a generic, universal method for processing XML documents (WEB_13, 2004). Most of the built parsers support DOM model

*Disadvantages:*

"Processing XML documents does not stop at parsing. The data from those documents then needs to be acted on. For most bussines applications, parsing is the first process step at all" (WEB_13, 2004). Some business organizations have found the DOM to be impractical for use in their business application environment due to its high execution cost, inefficient memory management and other implementation problems.

### 2.6.4 XML Technology Supports Existing Security Solutions

*Advantages:*

Since XML technology supports existing Internet and network infrastructure, it can take advantage of the increasing framework for providing security for these infrastructures (WEB_13, 2004). Supported security concepts are;

- *Authentication*, making sure receiver and sender are really who they say are.

- *Encryption*, protecting data; XML documents can be transferred over HTTP using SSL protocol.
- *Authorization*, rights to access data (read, write, change, execute); Techniques work just as well with XML as they do for other documents and protocols

*Disadvantages:*

XML supports existing security solutions but also introduces new security risks of its own that could pose serious threat to the integrity and exchange of data. XML documents may have reference to external DTDs (WEB_15, 2004). Any change that produces a fatal error in this DTD can then produce a chain reaction damaging XML processing. These changes may be;

- OPTIONAL typed can be changed to REQUIRED.
- Intruder can redefine default values for attribute (WEB_13, 2004)
- Intruder can manipulate entities so that they insert text into documents, it's possible to insert malicious content where an entity is used (WEB_13, 2004)

### 2.6.5 XML Is Not A Programming Language

XML technology is not a programming language. It is simply a document format which contains metadata. "This misalignment in understanding causes confusion not only about what XML is, but what it is capable of doing. XML require parsers and applications to process them" (WEB_13, 2004). XML itself is simple. However all the "action" really occurs in the next few layers of technology: schema validation, parsing, processing, integration, mapping, messaging and transformation. XML gives us a standard way to define a document format. That makes writing a parser easier. However, most businesses and industries will create their own versions of the language, which will then require more than just parsing to get an understanding of what that document means (WEB_13, 2004).

It is incorrect to call XML as a programming language, since there is no manner in which it instructs a computer how to process information, such as Delphi or C++ (WEB_13, 2004). XML is only a markup language not a programmatic language. Actually, some people have tried to replace scripting and programming languages with XML equivalents. But it has not been usable.

```
Function HelloWorld : Boolean;
  Begin
    Write('HelloWorld');
     Result := True;
  End;
```

```
<Function Name="HelloWorld"
          Type="Boolean">
   <call function="Write">
      <param type="String">
       Hello World
      </param>
   </call>
</Function>
```

XML is easy to learn, implement, read, and test. It has shortened product development time for most XML-related and data exchange projects like B2B. "XML is an effective, portable, easily customized data format that can easily sent over virtually any protocol" (WEB_13, 2004). While it is just a data format, it can be used for many different purposes, ranging from messaging to RPC.

The problem is that XML is being applied in every possible scenario even when it is not appropriate. This is primarily a problem in human nature in that people like to use new technologies for all problems.

## 2.7 XML Real Life Scenarios

Although XML technology has disadvantages, the number of projects which uses XML technology increases from day to day. Application areas of XML technology is below;

- Using XML documents as data sources
  - Processing documents using DOM model
  - Processing and presentation using XSL, XSLT (WEB_16, 2004) and XPath (WEB_17, 2004) technologies
- Using XML documents in the integration of different systems
  - B2B applications
    - XMLRPC, XMLHTTP (WEB_21, 2004)
  - Web Services
    - SOAP (WEB_18, 2004)
  - BizTalk
- Using XML documents to standardize GUI(Graphical User Interface)
  - UIML (User Interface Markup Language) (WEB_20, 2004)
  - XAML (XML Application Markup Language) (WEB_19, 2004)
- Using XML data as ActiveX component's content
- Using XML technology to define web resources

### 2.7.1 Using XML Documents As Data Sources

Tables in relational database management systems can easily be converted into XML documents. W3C developed DOM tree based model to process XML documents (WEB_14, 2004). There are lots of parsers which use DOM for processing XML documents. One of the most used XML parsers is Microsoft XML Parser (MSXML) (WEB_22, 2004). Following example shows an XML document, includes inventory information and code to process this XML document.

**Table 2.4 Use of XML Documents as DataBase**

```
<?xml version="1.0"?>
  <STOCKLIST>
    <STOCK>
      <CODE>01</CODE>
    </STOCK>
  </STOCKLIST>
```

```
(1)DomDoc:= CreateObject('MSXML.DOMDocument');
(2)DomDoc.LoadXML('<STOCKLIST> '
                + '  <STOCK> '
                + '    <CODE>01</CODE> '
                + '  </STOCK>'
                  ' </STOCKLIST>');
(3)kod := DomDoc.DocumentElement.

SelectSingleNode('STOCK').
  SelectSingleNode('CODE').Text;
```

Table 2.4 shows processing XML documents with MSXML DOM parser.

- (1) An instance of MSXML.DOMDocument (WEB_23, 2004) class which is named

  DomDoc is created.
- (2) XML data is loaded into DomDoc instance.
- (3) DomDoc is traversed to get the node value of the node "CODE"

Another two ways to process and view XML documents are using XSL and XPath technologies. XPath is used to traverse all nodes of an XML document (WEB_17, 2004). XSL is used to traverse all units of an XML document (elements, attributes, comment lines, processing instruction lines) and make all possible queries and conversions (WEB_16, 2004). XSLT allows us to convert an XML document into different document formats such as HTML, WML, PDF, CSV etc.

### 2.7.2 Use of XML Documents For Integration of Different Systems

One of the primary advantages of the XML technology is that it allows integration of different systems. In the past, some unformatted data types or EDI messages (WEB_24, 2004) were used for integration of different systems. Today, EDI messages still used in some sectors such as automotive and health.

Microsoft BizTalk server (WEB_25, 2004) is used for integration and communication between different organizations. It supports both XML and EDI technologies and allows conversion between these two technologies.

Building B2B applications with XMLHTTP and XMLRPC allow us to communicate different s ystems. For e xample, O nline B anking a nd D istribution A pplication t ransfer functions of Netsis Commercial Packet is built with XML technology. XMLHTTP technology is used to send XML messages from client to server over HTTP. After processing the received messages, server sends an XML message to the client.

**Table 2.5 XMLHTTP Technology**

**[Client]**

```
   (1)Set xmlhttp = CreateObject("Microsoft.XMLHTTP")
   (2)Set objXMLDoc = CreateObject("Microsoft.XMLDOM")
   (3)objXMLDoc.LoadXML ("<?xml version='1.0' ?>" & _
"<NETSIS_DAGITIM><></NETSIS_DAGITIM>")
   (4)xmlhttp.Open "POST", "http://www.netsis.com.tr/Dagitim", False
   (5)xmlhttp.Send objXMLDoc
   (7)msgbox xmlhttp.ResponseXML.DocumentElement.XML
```

**[Server]**

```
   Response.ContentType = "text/xml"
   (6)Response.Write (Request)
```

Table 2.5 shows use of XMLHTTP technology for XML messaging between client and server.

- (1) xmlhttp object of Microsoft.XMLHTTP class is created.
- (2) objXMLDoc object of MSXML.DOMDocument is created.
- (3) XML message is loaded into objXMLDoc
- (4) Connection established between client and server by using Open method
- (5) XML message is sent to the server by using Send method.
- (6) Server sends back the received XML message.
- (7) Client displays the received XML message.

SOAP applications and Web services take advantage of XML technology. SOAP allows us to access and activate objects on the servers by using of HTTP and XML technology. SOAP uses XML envelopes (WEB_18, 2004) to activate remote objects.

**Table 2.6 SOAP Message**

**[SOAP Header]**

```
POST /WebService1/Matematik.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/Topla"
```

**[SOAP Envelop]**

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Topla xmlns="http://tempuri.org/">
      <x>int</x>
      <y>int</y>
    </Topla>
  </soap:Body>
</soap:Envelope>
```

Table 2.6 shows SOAP message which activates *Topla* function of a Microsoft .Net Web Service *Matematik.asmx*. Message consists of two parts, a header and an envelope.

- Header includes service machine name, URL resource address and content type.
- Envelop includes structure and parameters of the “Topla” function in XML format

### 2.7.3 Use of XML Documents for Standardization of User Interfaces

If we want our applications to support different platforms such as pocket-pc, web or win32 we have to change the design of our application or we have to build different user interfaces for different platforms. For example, it is not easy to use all objects or styles of a win32 application in a web application. Because of these difficulties, organizations such as ISO and IEEE developed different standards and languages like UIML (User Interface Markup Language) (WEB_26, 2004).

These kinds of standardizations are insufficient because we also need a strong framework such as JVM (Java Virtual Machine) or .Net Framework. New operating system of Microsoft (LongHorn) Avalon (WEB_27, 2004) is used XAML to standardize user interfaces (XML Application Markup Language) (WEB_19, 2004).

**Table 2.7 XAML (WEB_27, 2004)**

```
<Canvas  xmlns="http://schemas.microsoft.com/2003/xaml" >
<Rectangle
   Fill="#33CC66"   Width="2in"  Height="1in"
   Canvas.Top="25" Canvas.Left="50"  StrokeThickness="6px"
   Stroke="Orange" />
<Ellipse
   Fill="yellow"
   CenterX="1.5in"
   CenterY="1.1in"
   RadiusX=".5in"
   RadiusY="1in"
   StrokeThickness="4px"
   Stroke="Blue" />
<Text
     Canvas.Top="50"  Canvas.Left="60"
     Foreground="#000000"  FontWeight="Bold"  FontFamily="Arial"
FontStyle="Normal"
FontSize="25">
   Hello Shapes!
</Text>
</Canvas>
```

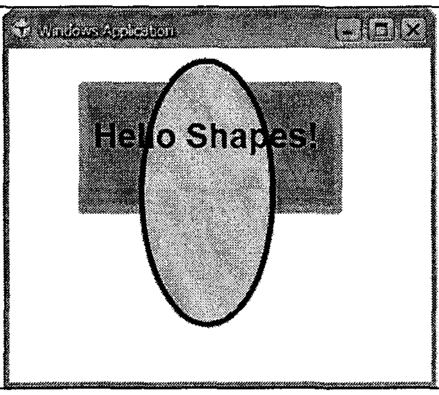


Table 2.7 shows a simple user interface developed in Avalon Operating System by XAML.

### 2.7.4 XML As The Structural Content Of ActiveX Components

Another application area of XML is the activation of ActiveX components. XML support of Microsoft Office Components (WEB_28, 2004) is a good example for such applications. Following example shows an XML code of a basic Excel application. You can also save an Excel document as an XML document by using *Save As* option of excel.

**Table 2.8 XML Data of An Excel Application**

```
<?xml version="1.0"?>
<Workbook
  xmlns="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:x="urn:schemas-microsoft-com:office:excel"
  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:html="http://www.w3.org/TR/REC-html40">
 ...
 <Styles>
  <Style/>
 </Styles>
 <Worksheet ss:Name="Sheet1">
 <Table>
   <Row>
     <Cell> <Data ss:Type="String">deneme</Data>  </Cell>
   </Row>
 </Table>
 </Worksheet>
</Workbook>
```

In Table 2.8, XML data of Excel application shows that the data in the first column of the first row of  Sheet1 is *String* and its value is *deneme.*

### 2.7.5 XML For Defining Web Resources

HTML deals with documents views more then their contents and Web is full of HTML documents. As a result Web became a collection of garbage. Because of this W3C developed RDF (WEB_29, 2004) to analyze resources and determine the relations.

RDF technology is more than a markup or formatting language but it is an API. Dom process XML documents by the help of the physical tree structure of XML documents but RDF is interested in the logical structure of an XML Document instead of its physical structure. Elements of RDF define some expressions about sources. On these expressions each resource can have more than one attribute. Each attribute include an attribute name and its value

**Table 2.9 RDF**

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description about="http://www.deu.edu.tr/BilMuh">
    <Author>
      Deniz KILINÇ
    </Author>
  </rdf:Description>
</rdf:RDF>
```

In Table 2.9 a web resource with the URL of http://www.deu.edu.tr/BilMuh. The author attribute of the resource is *Deniz KILINC.*

# CHAPTER THREE
# FIRST XML DOCUMENT

## 3.1 Introduction

This chapter tells how to create a basic XML document, examine document components, save the document, document display by the browser, and lastly how to format the document using a CSS style sheet on a browser.

## 3.2 Creation of XML Document

XML documents can be created in simple text editors like Microsoft Notepad. Of course tools like XML Spy (WEB_30, 2004), Macromedia Dreamweaver (WEB_31, 2004) can be used to create and arrange XML documents. First XML document example written in Notepad includes camera categories, and relative camera information. Readable special tags (<CAMERA>, <BRAND>, <MODEL>...) are used for categories and cameras. Even someone who does not know XML or web technologies can understand the content of this document by reading it.

**Table 3.1 First XML Document**

```
<?xml version="1.0" standalone="yes"?>
<CAMERA_CATEGORIES>
 <CATEGORY NAME="35MM">
  <CAMERA>
   <BRAND>Canon</BRAND>
   <MODEL>Z155</MODEL>
```

**Table 3.1 Continued...**

```
    <PRICE>1875000</PRICE>
    <WEIGHT>120 gr.</WEIGHT>
   </CAMERA>
   <CAMERA>
    <BRAND>Nikon</BRAND>
    <MODEL>S5Z</MODEL>
    <PRICE>1725000</PRICE>
    <WEIGHT>130 gr.</WEIGHT>
   </CAMERA>
  </CATEGORY>
  <CATEGORY NAME="DIGITAL">
   <CAMERA>
    <BRAND>Canon</BRAND>
    <MODEL>EOS 10D</MODEL>
    <PRICE>2249985</PRICE>
    <WEIGHT>135 gr.</WEIGHT>
   </CAMERA>
   <CAMERA>
    <BRAND>Minolta</BRAND>
    <MODEL>Dimage</MODEL>
    <PRICE>5999850</PRICE>
    <WEIGHT>110 gr.</WEIGHT>
   </CAMERA>
   <CAMERA>
    <BRAND>Nikon</BRAND>
    <MODEL>D100</MODEL>
    <PRICE>25499850</PRICE>
    <WEIGHT>140 gr.</WEIGHT>
   </CAMERA>
  </CATEGORY>
</CAMERA_CATEGORIES>
```

## 3.3 Processing Instruction (PI)

XML documents start with a line called processing instructions. Processing instructions start with <? and end with ?> characters. The first word after the leading characters is the name of the processing instruction. Instructions are passed to applications using XML documents by the processor, and technically, they are not counted as a component of the document (Harold E.R, 1999). Documents have the processing instruction given in the example above. The instruction has *version*, *standalone* and *encoding* attributes. Attributes are name-value pairs which have equals sign (=) between them. Value of an attribute is written in quotation marks after equals sign.

**Table 3.2 Processing Instruction Line**

| |
|---|
| **<?xml version="1.0" standalone="yes"?>** |

- *version* attribute is for telling XML parser which XML version published by W3C the document is compatible with this document. Now, there are two versions published by W3C which are version 1.0.
- s*tandalone* attribute tells whether the document is composed of only one document. An XML document may be composed of many XML documents and may reference other objects.

## 3.4 Elements

After the processing instruction line, element comes that is known as document element or root element. Elements are the basic units of XML documents (Harold E.R, 1999);

- An element, which is not an empty element, is composed of an opening tag, content information, and a closing tag.

**Table 3.3 Elements**

| *Opening Tag* | *Content Information* | *Closing Tag* |
|---|---|---|
| <BRAND> | Canon | </BRAND> |

- All tags include element type name.
- Element type names are placed between < > characters. Closing tag has / character before element type name.

- Element type name can start with letters, or _ underscore character, but not with numbers or - character.
- Element type names should not start with xml, XML, Xml, XMl, xML, xmL, xMl.
- Element type names do not contain spaces.

**Table 3.4 Element syntax**

| *Wrong Syntax* | *Wrong Syntax* |
|---|---|
| <BRAND>Canon<bRANd> | <BRAND>Canon<brand> |

- Elements must be nested properly.

**Table 3.5 Root Element and Sub-Elements**

```
<CAMERA_CATEGORIES>
  <CATEGORI NAME="35MM">
    <CAMERA>      ....      </CAMERA>
     ....
  </CATEGORY>
  <CATEGORY NAME="DIGITAL">
     ....
  </CATEGORY>
    ....
</CAMERA_CATEGORIES>
```

In the example, *CAMERA_CATEGORIES* element is the root element, and has two sub-elements. *CATEGORY sub-*element has *CAMERA* sub-elements, which have *BRAND, MODEL, PRICE, WEIGHT* child elements. Child elements include content data. CAMERA sub-element is the parent element of child elements.

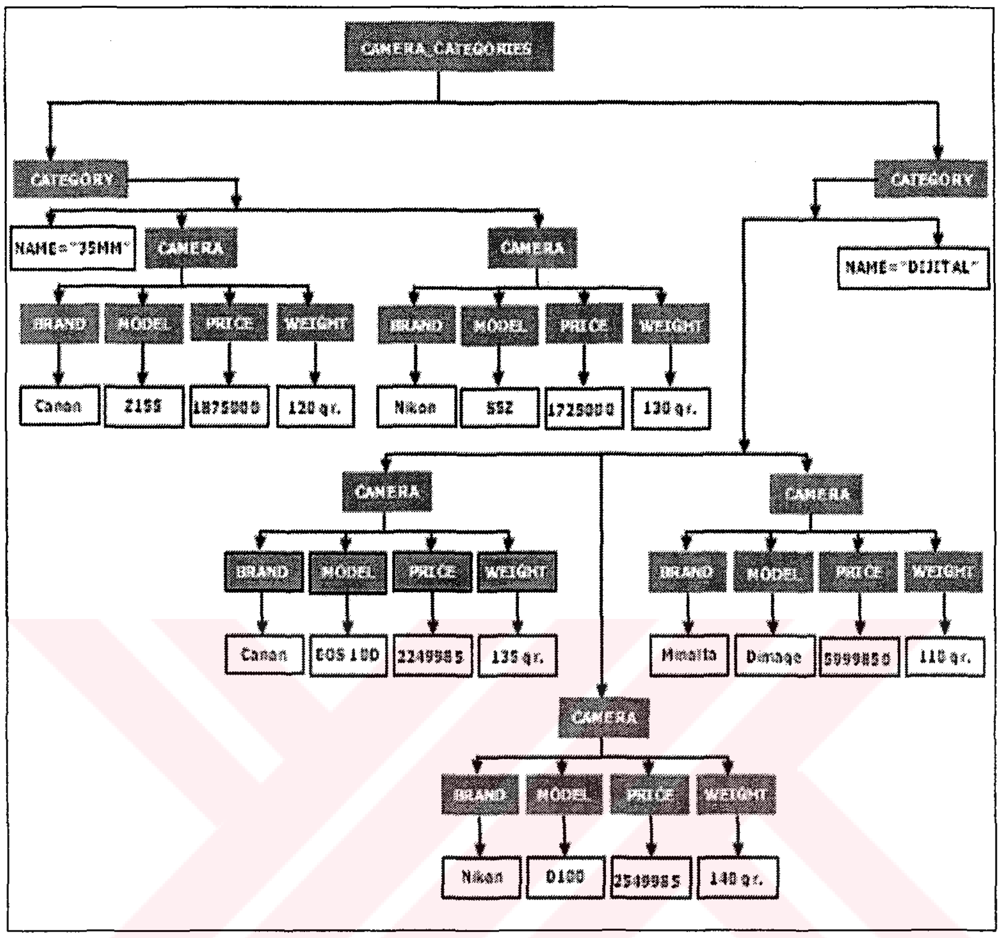
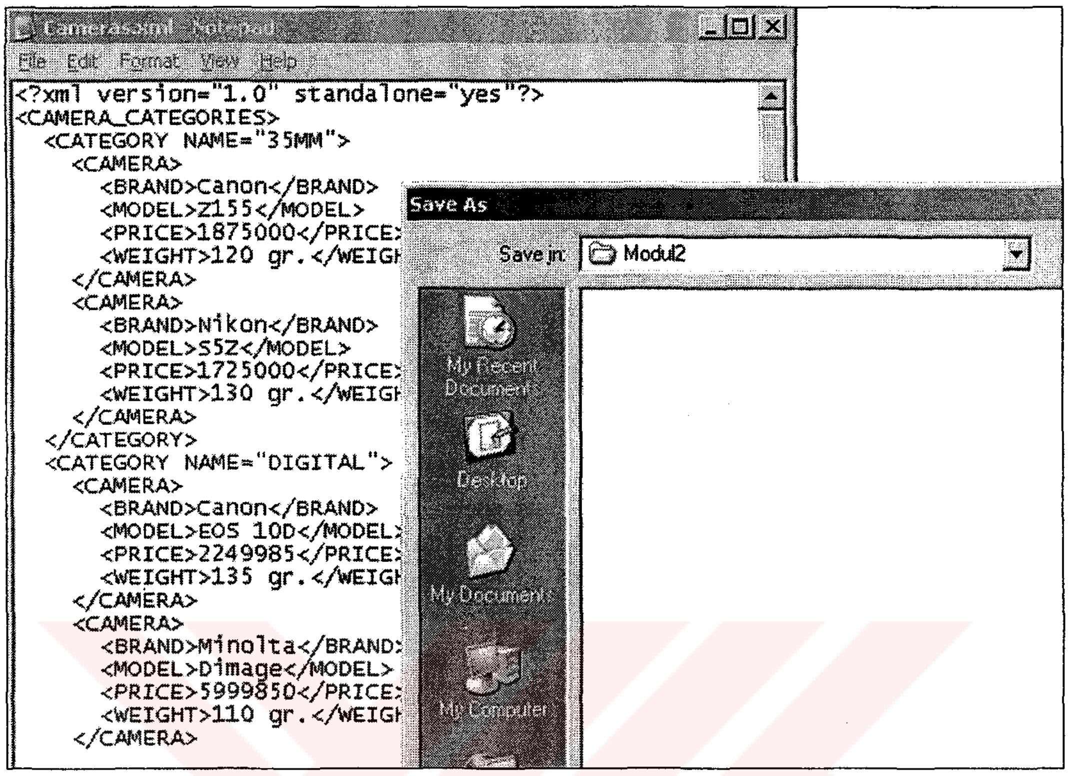


**Figure 3.1 Camera categories tree hierarchy**

## 3.5 Saving an XML Document

After writing XML document, we must save the document such that its save as type will be xml (first.xml, Cameras.xml). If an editor like Microsoft Word is used, document must be saved as plain text. Figure 3.2 shows how to save the document written in notepad as *Cameras.xml*.

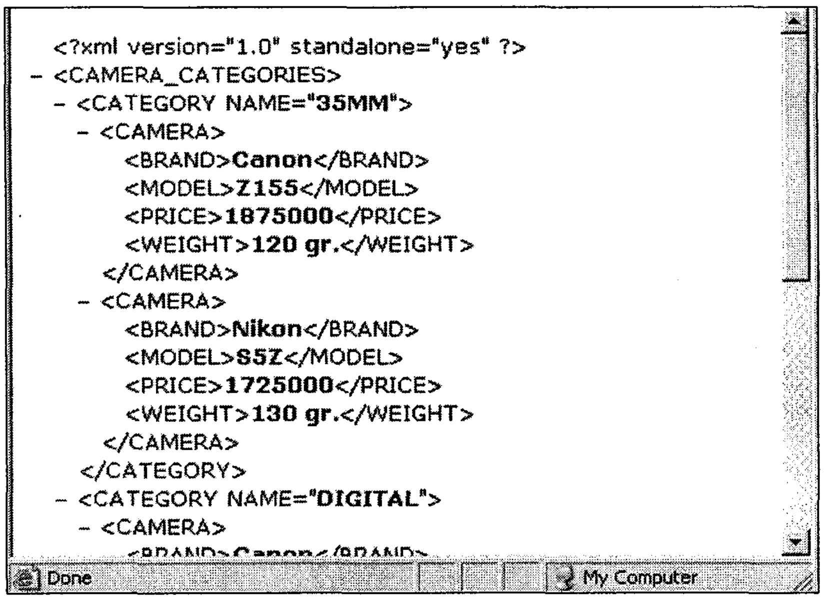


```
<?xml version="1.0" standalone="yes"?>
<CAMERA_CATEGORIES>
  <CATEGORY NAME="35MM">
    <CAMERA>
      <BRAND>Canon</BRAND>
      <MODEL>Z155</MODEL>
      <PRICE>1875000</PRICE:
      <WEIGHT>120 gr.</WEIGH
    </CAMERA>
    <CAMERA>
      <BRAND>Nikon</BRAND>
      <MODEL>S5Z</MODEL>
      <PRICE>1725000</PRICE:
      <WEIGHT>130 gr.</WEIGH
    </CAMERA>
  </CATEGORY>
  <CATEGORY NAME="DIGITAL">
    <CAMERA>
      <BRAND>Canon</BRAND>
      <MODEL>EOS 10D</MODEL:
      <PRICE>2249985</PRICE:
      <WEIGHT>135 gr.</WEIGH
    </CAMERA>
    <CAMERA>
      <BRAND>Minolta</BRAND:
      <MODEL>Dimage</MODEL>
      <PRICE>5999850</PRICE:
      <WEIGHT>110 gr.</WEIGH
    </CAMERA>
```

**Figure 3.2 Saving XML documents**

## 3.6 Displaying XML Document by a Browser

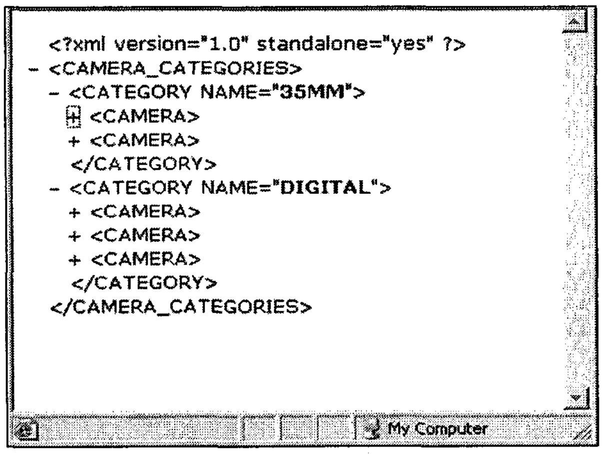
XML documents, just like HTML pages, can be displayed in Internet Explorer which is a web browser. Hence tags in HTML pages are predefined standard tag set, web browser knows a tag's style. For example, an HTML browser reading *<B>* tag, makes the following text bold. Since elements in XML documents are defined by developers, they are recognized only as character data by Internet Explorer. Internet Explorer displays XML elements in a properly nested, and a treeview fashion as colored.



```
  <?xml version="1.0" standalone="yes" ?>
- <CAMERA_CATEGORIES>
  - <CATEGORY NAME="35MM">
    - <CAMERA>
        <BRAND>Canon</BRAND>
        <MODEL>Z155</MODEL>
        <PRICE>1875000</PRICE>
        <WEIGHT>120 gr.</WEIGHT>
      </CAMERA>
    - <CAMERA>
        <BRAND>Nikon</BRAND>
        <MODEL>S5Z</MODEL>
        <PRICE>1725000</PRICE>
        <WEIGHT>130 gr.</WEIGHT>
      </CAMERA>
    </CATEGORY>
  - <CATEGORY NAME="DIGITAL">
    - <CAMERA>
```

Figure 3.3 Viewing XML documents via Internet Explorer

Root and sub-elements can be expanded and collapsed. So elements can be identified and logical structure of document can be grasped easily. Clicking + character near the elements expand the collapsed sub-elements. Clicking - character near the elements collapses the expanded sub-elements. Figure 3.4 shows how collapsed Canon camera under the 35MM camera category is expanded by clicking.



```
  <?xml version="1.0" standalone="yes" ?>
- <CAMERA_CATEGORIES>
  - <CATEGORY NAME="35MM">
    + <CAMERA>
    + <CAMERA>
    </CATEGORY>
  - <CATEGORY NAME="DIGITAL">
    + <CAMERA>
    + <CAMERA>
    + <CAMERA>
    </CATEGORY>
  </CAMERA_CATEGORIES>
```

Figure 3.4 Expanding and collapsing XML documents

## 3.7 Displaying XML Document by Using Cascade Style Sheet (CSS)

Since XML document is composed of tags defined by developer, it does not recognized by the web browser as a document, style and format. So, a style sheet, which shows the format of document for web browser, should be defined with the document. Cascade style sheet (CSS) (WEB_32, 2004) is a kind of style sheet. CSS is originally designed for formatting elements of HTML pages. CSS provides the creation of font family, size, and type. CSS is also developed by W3C (WEB_5, 2004) like XML. Since HTML developers are used to CSS, it has been a great advantage in its usage with XML.

## 3.8 Creation of CSS

Create a new file in a text editor; write the CSS content in table 3.6, save the file as Cameras.css.

**Table 3.6 Cameras.css Style Sheet (WEB_32, 2004)**

```
CATEGORY
{
 Font-Family        : Verdana;
 Font-Weight        : Bold;
 Font-Size          : 14pt;
 Color              : DarkBlue;
 Border-Style       : Outset;
 Border-Width       : 4;
 Display            : Block;
 Text-Align         : Center
}
CAMERA
{
 Font-Family        : Times New Roman;
 Font-Weight        : Normal;
 Font-Size          : 12pt;
 Color              : Blue;
 Border-Style       : Groove;
 Border-Width       : 1;
 Display            : Block;
```

**Table 3.6 Continued...**

```
 Text-Align        : Left;
 Text-Decoration   : Underline;
 Margin-Left       : 10px;
}
```

CSS in table 3.6 is written to format CATEGORY AND CAMERA tags.

- *Font-Family;* which font family is used, *Font-Weight;* whether the font is bold, normal or italic,
- *Font-Size;* size of the font, *Color;* color of text,
- *Border-Style;* style of borders, *Border-Width;* width of borders,
- *Display;* display of text,
- *Text-Align;* whether text is displayed left-aligned, centered, or right-aligned,
- *Text-Decoration;* whether subscript or superscript is used,
- *Margin-Left;* left margin, *Margin-Bottom;* right margin.

## 3.9 Integrating CSS Style Sheet with XML Document

After forming XML document and CSS style sheet, the browser should know how to integrate XML document with CSS. Thus, a new processing instruction which declares the style sheet of XML document must be added. PI line is written as *<?xml-stylesheet?>* ,and has two attributes;

- *type* attribute shows language of style sheet,
- *href attribute shows URL of style sheet.*

Open Cameras.xml document in table 3.1 with Notepad, and add the following instruction as the second processing instruction to the document.

**Table 3.7 Cameras.css processing instruction**

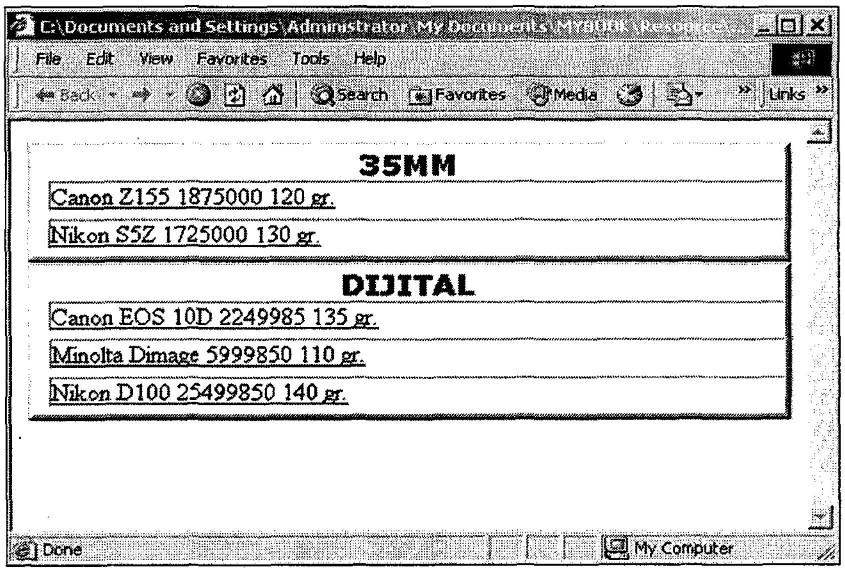
```
<?xml-stylesheet type="text/css" href="Cameras.css"?>
```

For easy formatting of document by Cameras.css, NAME attributes of CATEGORY tags should be transformed into <NAME> child tags. After adding new PI line and the transformation, XML document is saved as *"CamerasCss1.xml"*.

**Table 3.8 First XML Document**

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/css" href="Cameras.css"?>
<CAMERA_CATEGORIES>
 <CATEGORY>
  <NAME>35MM</NAME>
  <CAMERA>...</CAMERA> <CAMERA>...</CAMERA>
 </CATEGORY>
 <CATEGORY>
  <NAME>DIJITAL</NAME>
  <CAMERA>...</CAMERA> <CAMERA>...</CAMERA>
<CAMERA>...</CAMERA>
 </CATEGORY>
</CAMERA_CATEGORIES>
```

The rendered view of XML document by the browser is shown in Figure 3.5.


C:\Documents and Settings\Administrator\My Documents\MYBOOK\Resource...
File Edit View Favorites Tools Help
Back Search Favorites Media Links
Done
My Computer

35MM
Canon Z155 1875000 120 gr.
Nikon S5Z 1725000 130 gr.
DIJITAL
Canon EOS 10D 2249985 135 gr.
Minolta Dimage 5999850 110 gr.
Nikon D100 25499850 140 gr.

**Figure 3.5 Display of XML Document by using CSS style sheet**

# RESEARCHING XML AND RELATED TECHNOLOGIES

by

Deniz KILINÇ

June, 2004

İZMİR

# CHAPTER FOUR
# RESEARCHING XML 1.0 EBNF RULES

## 4.1 BNF Grammar

The main reason of XML's much popularity according to its ancestor SGML, is its simplicity and usability for web applications. This simplicity and usability advantage comes from W3C's compact XML specification framework, with other words, from compact EBNF grammar (WEB_33, 2004).

BNF is a shortcut for Backus-Naur-Form. BNF grammars are an outgrowth of compiler theory (WEB_10, 2004). "A BNF grammar defines what is and is not a syntactically correct or not. It is possible to compare a document to a BNF grammar and determine precisely whether it does or does not meet the conditions of that grammar" (Harold E.R, 1999). BNF grammar has three parts;

- *Terminals;* some special characters and words like <, >, ]]>, CDATA, #REQUIRED in XML grammars.
- *Non-terminals;* Elements which will be converted to terminals by using rules. Document element of XML starts with first EBNF rule, and then converted to terminals.
- *Productions-Rules;* Mappings from non-terminals to non-terminals or other sequences of terminals.

For example, we assume a BNF grammar for *Turkish language words,* start with vowel, followed by one or more consonants, ends with a vowel. The solution of the problem is below;

- Three rules are defined for grammar
- *TrueWord, Vowel* and *Consonant* words are non-terminals
- *aeıioöuü* letters are terminals

```
[1] TrueWord        ::= Vowel     Consonant+     Vowel
[2] Vowel           ::= [a e ı i o ö u ü]
[3] Consonant       ::= [^ a e ı i o ö u ü]
```

If *arti* word is controlled for BNF grammar, firstly non-terminal first rule is processed. According to rule *TrueWord* must be start with Vowel. Then Vowel non-terminal is searched and second rule is processed. It tells us Vowel non-terminal includes all Turkish vowel letters. Because first letter of *arti* is a vowel, processing continues for other letters. First rule is processed again and it says one or more Consonant must be occurred. Consonant non-terminal is searched and third rule is processed. It tells us Consonant non-terminal includes all Turkish consonant letters.

Because, second and third letters of *arti* are consonants processing continues for the last letter. First rule is processed again and encountered with last non-terminal Vowel. second rule is processed again. Since, last letter of *arti* fits to Vowel rule, processing is finished. As a result *arti* word fits to the BNF grammar. Some suitable words for grammar are atkı, etki, içki, arsa, atklkdbfbi, ehjhjhi and some not are katkı, etek, aidat, artkgiklw.

## 4.2 Basic BNF

BNF grammar has special characters and rules. Characters are used to create rules and rules are used to create grammar.

**Table 4.1 Basic BNF Symbols and Characters (Harold E.R, 1999)**

| Symbol | Description |
|---|---|
| #xN | N is a hexadecimal integer, and #xN is the Unicode character with the number N |
| [a-z A-Z] | Matches character range a-z or A-Z |
| [#xN-#xN] | matches any character in the specified range where N is the hexadecimal value of a Unicode character |
| [^a-z] | matches any character not in the specified range |
| [^abc] | matches any character not in the list |
| A* | Zero or more occurrences of A |
| A+ | One or more occurrences of A |
| A? | Zero or one occurrences of A |
| A B | A comes after B |
| A \| B | Matches A or B but not both |
| A – B | matches any string that matches A and does not match B |
| ^A | Matches except A |
| (A* B+) \| (C?) | (Zero or more A and one or more B) Or zero or one C |
| /* Comment */ | Comment lines |
| [WFC: ] | Well-formedness constraint associated with this production that documents must meet in order to qualify as well-formed. Well-formedness constraints will be found in the specification, but are not encapsulated in the BNF grammar. |
| [VC: ] | Validity constraint associated with this production that documents must meet in order to qualify as valid. Validity constraints will be found in the specification, but are not encapsulated in the BNF grammar. |

XML 1.0 version has 89 EBNF rules (WEB_10, 2004). EBNF (Extended-BNF) grammar is the next version of BNF. Technically, EBNF has some extra capabilities which are not supported by conventional, compiler based BNF.

## 4.3 Well-Formed Documents

"A data object is an *XML document* if it is well-formed" (WEB_12, 2004). Document parsing depends on well-formedness. "A well-formed XML document may in addition be valid if it meets certain further constraints" (WEB_12, 2004).

## 4.4 Documents

An XML document b ecomes physical and logical structures. Document's p hysical structure is a set of units called entities. Entities can reference each other. XML document starts with "document" entity. Document's logical structure includes elements, character references, comments and processing instructions.

**Table 4.2 Document Rule (WEB_12, 2004)**

| **Document** | | |
|---|---|---|
| **[1] document** | ::= | prolog element Misc* |

The first rule has 3 non-terminal elements.

1. According to rule, an XML document must start *prolog* element. Simply, a prolog element includes DTD and XML declaration definitions.
2. *A root element must follow the Prolog* element. Elements have start- and end-tags, and may have character data, other elements, or both between these tags. Empty elements may use an empty-element tag instead of a start- and end-tag pair(<emptyelement attrib="1"/>). There is exactly one element, called the *root*, or document element, no part of which appears in the content of any other element. If an <X> element has <Y> element as a child, <X> element is the parent of <Y> element. With other words, <Y> element is the child of <X> element.

3. Third non-terminal "*Misc*" element is optional. *Misc* element includes white-spaces, comments or processing instructions.

If an XML document fits the first EBNF rule, then document is called as "well-formed document" (WEB_12, 2004).

**Table 4.3 Well-formed XML document**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<!-- DTD declaration-->
<ROOT_ELEMENT>
 <PARENT>
  <CHILD> Content Information</CHILD>
 </PARENT>
</ROOT_ELEMENT>
<?Processing Instruction1?>
<?Processing Instruction2?>
```

According to rule, an XML document includes only one root element. If an XML document has more than one root element, it is not well-formed.

**Table 4.4 Non-well formed XML document**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<!-- DTD declaration-->
<ROOT_ELEMENT>abc </ROOT_ELEMENT>
<ROOT_ELEMENT>def </ROOT_ELEMENT>
<?Processing Information?>
```

## 4.5 Characters

The ISO/IEC 10646 standard created by a commission of the ISO and the International Electrotechnical Commission in 1993 specifies the Universal Multiple-Octet Coded Character Set (UCS). The Universal Character Set is a collection of characters (usually, elements of alphabets, numeric digits, and other characters such as punctuation) that aims to represent all the written languages of the world (WEB_12, 2004).

An octet is a grouping of eight bits of information. An octet can represent 256 different values. This is enough for all the characters on an English-language keyboard and some other miscellaneous ones, but certainly not enough to cover all the characters in all the languages that people want to use when storing documents on computers. Doing this requires multiple octets for each character.

Using two octets per character, you can represent 65,536 different characters; the ISO 10646 version of this is known as UCS-2. Four octets, UCS-4, can represent over two billion different characters (of the 32 bits in the four octets of a UCS-4 character, the first must be "0", leaving over two billion possible combinations of the remaining thirty-one bits) (WEB_12, 2004).

Unicode is a standard developed by the Unicode Consortium for representing characters with 16 bits. This group is a separate organization from the ISO. "These two standards, in order to remain backward-compatible with existing text files, have the same first 128 characters as the 128 characters in the ASCII character set" (WEB_12, 2004).

XML s upports both UCS-2 and Unicode standards. In addition to these standards, UTF-8 and UTF-16 (UTF, UCS Transformation Format) are also supported. Character ranges of XML documents are declared in 2nd EBNF rule.

**Table 4.5 Character Range Rule (WEB_12, 2004)**

| **Character Range** | | |
|---|---|---|
| **[2] Char** | ::= | #x9 \| #xA \| #xD \| [#x20-#xD7FF] \| [#xE000-#xFFFD] \| [#x10000-#x10FFFF] |

#x at the beginning of a number shows that it's written in hexadecimal, or base 16 notation, as opposed to the decimal, *base 10* notation that non-programmers are accustomed to. Hexadecimal notation represents the decimal notation numbers 10 through 15 using the letters. For example, 9 and 10 numbers are mapped to #x9 and #xA. According to 2nd rule, characters in the range of 9, 10, 13, 32 – 55295, 57344 – 65533, 65536 – 1114111 can be use as XML characters. Using hexadecimal character system is more efficient than *base 10* system.

## 4.6 Common Syntactic Constructs

In XML EBNF grammar, there exist mostly used characters. White-space characters are in this category.

**Table 4.6 White Space Rule (WEB_12, 2004)**

| White Space |
|---|
| [3] S ::= (#x20 \| #x9 \| #xD \| #xA)+ |

By space (#x20) characters, it means ASCII character 32—the character you type by pressing your keyboard's space bar. #x9 is the character that you type with your Tab key. Carriage returns and line feeds are two different characters.

**Table 4.7 Names and Tokens Rule (WEB_12, 2004)**

| Names and Tokens | | |
|---|---|---|
| **[4] NameChar** | ::= | Letter \| Digit \| '.' \| '-' \| '_' \| ':' \| CombiningChar \| Extender |

Characters are specified as letters, digits and other characters. Letters can be alphabetic simple letters or combinations of more than one element. For example, ğ character in Turkish alphabet, is the combination of g and ˘ characters.

**Table 4.8 Names and Tokens (WEB_12, 2004)**

| **Names and Tokens** | | |
|---|---|---|
| **[5] Name** | ::= | (Letter \| '_' \| ':') (NameChar)* |
| **[6] Names** | ::= | Name (S Name)* |

Names are very important units or concepts for XML documents. "Tokens or non-terminal elements are atomic units of XML documents" (WEB_12, 2004). Tokens are integrated according to XML EBNF rules and non-terminal elements are becomes, whit the non-terminal elements.

**Table 4.9 Valid and non-valid XML names**

| **Valid XML names** | **Non-valid XML names** |
|---|---|
| <Book> | <Book 1> |
| <Book1> | <.Book> |
| <_Book.1> | <1.Book> |
| <_1.Book> | <Book!> |
| <:Book> | <-Book> |
| <______> | <Book,1> |

Element Type Name can although start with any combination of XML word, (('X'|'x') ('M'|'m') ('L'|'l')); XML, xML, xmL, xml, XmL, Xml, XMl and xMl words are reserved for future use. An xml element can be named as <xmlelement>, but if W3C decides to create a new element using this name, applications using <xmlelement> must be changed.

Namespaces are special XML units to specify element type names. For example, if there exist two DTDs with the names *type1.dtd* and *type2.dtd*, and each of DTD uses the element <common>, namespaces must be used to separate logic of documents. (*<type1:common> <type2:common>*). *Name Tokens* are similar to element type names.

**Table 4.10 Names and Tokens Rule (WEB_12, 2004)**

| Names and Tokens | | |
|---|---|---|
| [7] **Nmtoken** | ::= | (NameChar)+ |
| [8] **Nmtokens** | ::= | Nmtoken (S Nmtoken)* |

Nmtokens can be constructed using NameChars. Nmtoken can be start with a letter, number and an underscore. They must not start with a letter like element type names (WEB_12, 2004). If address information wants to be kept in an XML document, the type of this element or attribute must be considered deeply. Since, NMTOKEN type can not contain special characters and the type of attribute must be set as CDATA.

**Table 4.11 Attribute in NMTOKEN and CDATA types**

| |
|---|
| <!ATTLIST personnel address **NMTOKEN** #REQUIRED> |
| <!ATTLIST personnel address **CDATA** #REQUIRED> |

**Table 4.12 Valid and non-valid Nmtokens**

| Valid Nmtokens | Non-valid Nmtokens |
|---|---|
| Book | Book 1 |
| Book 1 | Book,1 |
| _Book.1 | Book! |
| _1. Book | (Book) |
| : Book | # Book |
| ______ | Kitap$1 |
| - Book | |
| 1. Book | |

Entities are special XML units which provide to define and reference audio, mpeg like binary and unparsed files. The values of entities are shown between double or single quotes.

**Table 4.13 Literals Rule (WEB_12, 2004)**

| Literals |
|---|
| [9] **EntityValue** ::= '"' ([^%&"] \| PEReference \| Reference)* '"' <br> \| "'" ([^%&'] \| PEReference \| Reference)* "'" |

Values between single or double quotes are named “Literal Data”. According to 9th EBNF rule, entities can not include “%” and “&” characters.

**Table 4.14 Entity usage**

| |
|---|
| <!**ENTITY** DEU 'Dokuz Eylul University'> |

**Table 4.15 Valid and non-valid entities**

| Valid Entities | Non-valid Entities |
|---|---|
| “Dokuz Eylul” | “Dokuz Eylul’ |
| ‘Dokuz Eylul‘ | ‘Dokuz Eylul” |
| “Deniz &amp; Sibel ” | “Deniz & Sibel ” |
| “H2O &#37;80 i ” | “H2O %80 i ” |

Attribute values are also located between single and double quotes. They can include all values except <, & and “ characters.

**Table 4.16 Literals Rule (2) (WEB_12, 2004)**

| Literals | | |
|---|---|---|
| **[10] AttValue** | ::= | '"' ([^<&"] \| Reference)* '"' |
| | | \| "'" ([^<&'] \| Reference)* "'" |

**Table 4.17 Valid and non-valid attributes**

| Valid Attributes | Non-valid Attributes |
|---|---|
| “Value” | “Value’ |
| ‘Value‘ | ‘Value” |
| “Deniz &amp; Sibel ” | “Deniz & Sibel ” |
| “H2O %80 i ” | “H2O %80 i ” |
| “x lt; y” | “x < y” |

SystemLiterals are not recognized as markup and are not parsed. So, characters like &, <, > and % can be assigned as SystemLiteral value. Values are located between single or double quotes.

**Table 4.18 Literals Rule (3) (WEB_12, 2004)**

| Literals | | |
|---|---|---|
| **[11] SystemLiteral** | ::= | ('"' [^"]* '"') \| ("'" [^']* "'") |

**Table 4.19 Valid and Non-Valid SystemLiterals**

| Valid SystemLiterals | Non-Valid SystemLiterals |
|---|---|
| "Value" | "Value' |
| 'Value !' | 'Value" |
| "Deniz <and/> Sibel " | "Deniz"s Book" |
| "H2O %80 i " | 'Deniz's Book' |
| "Deniz's Book" | |

Public Id Literal values are the combination of PubidChars and PubidChars include white-spaces, a-z and A-Z letter range, 0-9 digit range and characters - *'()+,./:=?; !*# @ $*_%.

**Table 4.20 Literals Rule (4) (WEB_12, 2004)**

| Literals | | |
|---|---|---|
| **[12] PubidLiteral** | ::= | '"' PubidChar* '"' \| "'" (PubidChar - "'")* "'" |
| **[13] PubidChar** | ::= | #x20 \| #xD \| #xA \| [a-zA-Z0-9] \| [-'()+,./:=?;!*#@$_%] |

## 4.7 Character Data and Markup

XML markup has the format of start tags, end tags, empty element tags, entity references, character references, comments, CDATA sections, document type declarations (DTD) and processing instructions (PI). "Data except XML markup is called character data" (WEB_12, 2004).

- *Start and end tags;* <BRAND>Canon</BRAND> shows start and end tags together
- *Empty elements;* <BRAND NAME='Canon'/>
- *Entity references;* <!ENTITY DEU 'Dokuz Eylul University'> and "&gt;"
- *Character references; &#x50*

- *Comments;* <!-- CAMERA BRAND ELEMENT-->
- *CDATA sections;* Data in this section is cancelled by XML parser. <![CDATA[<BRAND NAME='Canon'/>]]>, in this example <BRAND> is not recognized as an XML element. It is only simple text.
- *Document type declarations;* Elements, attributes and entities are defined in DTDs. <!DOCTYPE category SYSTEM 'category.dtd'>
- *Processing instructions;* Used for to transfer data to applications using XML processors. <?xml version='1.0' standalone='yes'?>

**Table 4.21 Character Data Rule (WEB_12, 2004)**

| **Character Data** | | |
|---|---|---|
| **[14] CharData** | ::= | [^<&]* - ([^<&]* ']]>' [^<&]*) |

CharData can not include <, & characters and ]]> CDATA closing sections. These characters can be used within PIs, comments and CDATA sections.

## 4.8 Comments

"*Comments* may appear anywhere in a document outside other markup; in addition, they may appear within the document type declaration at places allowed by the grammar" (WEB_12, 2004). Comment lines are generally used to put some explanation to XML code line.

**Table 4.22 Comments Rule (WEB_12, 2004)**

| **Comments** | | |
|---|---|---|
| **[15] Comment** | ::= | '<!--' ((Char - '-') \| ('-' (Char - '-')))* '-->' |

Comment element starts with the <!-- string, then all characters except -- can follow 2nd Char rule. Finally, it is finished with --> string.

**Table 4.23 Valid and non-valid comments**

| Valid comments | Non-valid Comments |
|---|---|
| <!-- Camera --> | <!--Deniz -- Sibel --> |
| <!-- & Camera & --> | <!--Deniz -- Sibel -->> |
| <!-- Deniz <and/> Sibel --> | |
| <!-- Deniz ........ <and/> Sibel --> | |

### 4.9 Processing Instructions

"Processing instructions are used to transfer data to applications which use XML documents" (WEB_12, 2004). PI lines are also not recognized as XML character data like XML comments. They start with <? literal and ends with ?> string. PITarget is the name of processing instruction. These PI names can be got the all values except the XML word's upper and lower case combinations.

**Table 4.24 Processing Instructions Rule (WEB_12, 2004)**

| Processing Instructions | | |
|---|---|---|
| **[16] PI** | ::= | '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>' |
| **[17] PITarget** | ::= | Name - (('X' \| 'x') ('M' \| 'm') ('L' \| 'l')) |

PI element starts with *<?* string, then PITarget element, white-space character and characters specified in 2nd rule except *?>* string follows, finally it is finished with *?>* string.

**Table 4.25 Valid and Non-valid PIs**

| Valid PIs | Non-valid PIs |
|---|---|
| <?delphi version="6.0" param1="X"?> | <? delphi version="6.0" param1="X"?> |
| <?PI for a delphi application?> | <?delphi sample!> |
| <?xml-stylesheet type="text/css" href="Cameras.css"?> | <?xml PI line?> |

## 4.10 CDATA Sections

CDATA section is some text to be identified that should escape parsing. In other words, if there's anything in that text that would normally be considered as XML markup, treat it as character data. CDATA sections begin with the string *<![CDATA[* and end with the string *]]>*.CDATA sections are popular for showing demonstration XML, HTML or SGML markup within an XML document.

**Table 4.26 CDATA Sections (WEB_12, 2004)**

| **CDATA Sections** | | |
|---|---|---|
| **[18] CDSect** | ::= | CDStart CData CDEnd |
| **[19] CDStart** | ::= | '<![CDATA[' |
| **[20] CData** | ::= | (Char* - (Char* ']]>' Char*)) |
| **[21] CDEnd** | ::= | ']]>' |

In table 4.27, it is shown that how HTML elements can be hold within other HTML fixed markup. XML parser does not process these lines and recognized as character data not markup.

**Table 4.27 CDATA Usage**

```
<HTMLLESSON>
  <![CDATA[
            <HTML>
            <BODY>
              <P><B>First lesson about HTML technology...</B></P>
            </BODY>
            </HTML>
   ]]>
</HTMLLESSON>
```

**Table 4.28 Valid and non-valid CDATA Sections**

| Valid CDATA Sections | Non-valid CDATA Sections |
|---|---|
| <![CDATA[ <sample> &/......]]> | <![CDATA[ data ]]>.....]]> |
| <XMLDERSI><br><![CDATA[<br>CDATA sections start with <![CDATA[<br>string<br>]]><br></XMLDERSI> | |

## 4.11 Prolog and Document Type Definitions

"XML documents may, and should, begin with an XML declaration which specifies the version of XML being used" (WEB_12, 2004). This section of the XML specification describes markup that can make a document even more useful, because it provides extra information to a processing program about the document and its structure.

**Table 4.29 Prolog Rule (WEB_12, 2004)**

| Prolog | | |
|---|---|---|
| **[22] prolog** | ::= | XMLDecl? Misc* (doctypedecl Misc*)? |

Prolog element starts with an *XMLDecl* element, then an optional *Misc* element, a *doctypedecl* element and an optional Misc follows.

**Table 4.30 Valid and non-valid Prologs**

| Valid Prologs | Non-valid Prologs |
|---|---|
| <?xml version="1.0"?> | <?xml version="1.0"?> |
| <?xml version="1.0"?><br><!—Comment Line--> | <!-- Comment Line--><br><?xml version="1.0"?> |
| <?xml version="1.0" standalone="yes"?><br><?xml-stylesheet type="text/css" href="Cameras.css"?> | |
| <?xml version="1.0"?><br><!--Comment Line--><br><?xml-stylesheet type="text/css" href="Cameras.css"?> | <?xml-stylesheet type="text/css" href="Cameras.css"?><br><?xml version="1.0"?><br><!--Comment Line--> |

**Table 4.31 Prolog Rule (2) (WEB_12, 2004)**

| Prolog | | |
|---|---|---|
| **[23] XMLDecl** | ::= | '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>' |

XML declaration line starts with *<?xml* string, then follows with an *VersionInfo* element, then follows an optional EncodingDecl element, an optional SDDecl element and finally a white-space character.

**Table 4.32 Valid and non-valid XML declarations**

| Valid XML declarations | Non-valid XML declarations |
|---|---|
| <?xml version="1.0"?> | <?xml ?> |
| <?xml version="1.0" encoding="utf-8"?> | <?xml standalone="no"?> |
| <?xml version="1.0" encoding="utf-8" standalone="yes"?> | <?xml encoding="utf-8"?> |
| <?xml version="1.0" standalone="no"?> | <?xml version="1.0"??> |

VersionInfo element starts with white-space character, then follows fixed *version* keyword, an *Eq* element which is = sign. There can be a space at the left and right of sign. Finally, VersionInfo element ends with *VersionNum* element within single or double quotes.

**Table 4.33 Prolog Rule (3) (WEB_12, 2004)**

| Prolog | | |
|---|---|---|
| **[24] VersionInfo** | ::= | S 'version' Eq ('VersionNum' \| " VersionNum") |
| **[25] Eq** | ::= | S? '=' S? |
| **[26] VersionNum** | ::= | ([a-zA-Z0-9_.:] \| '-')+ |

VersionNum element can include letters in the range of a-z and A-Z, characters like ., :, _ and digits between 0-9.

**Table 4.34 Valid and non-valid XML versions**

| Valid XML versions | Non-valid XML versions |
|---|---|
| Version=”1.0” | Version=’1.0” |
| Version=’1.0’ | Version=”1.0’ |
| Version = ”1.0” | ”1.0”= version |
| Version=”1.a” | Version=”v1,6” |
| Version=”1.2.1” | Version=”1 6” |
| Version=”1.2RE4” | Version=”1 . 2RE4” |
| Version= “ver1.0_L” | Version=”ver 1.0_L” |

**Table 4.35 Prolog Rule (4) (WEB_12, 2004)**

| Prolog |
|---|
| **[27] Misc** ::= Comment \| PI \| S |

Misc element can be in the form of comment line, processing instruction line or white space character. It is a non-terminal element in the first *Document* rule.

Document type definition is a mechanism that specifies XML document’s logical structure (WEB_34, 2004). Definitions are saved in files which has DTD extensions. These files are declarations for elements, attributes, entities and external reference definitions. An XML document which includes DTD file or references to a DTD file is formed and named as valid document.

**Table 4.36 XML document that references external entities**

```
<?xml version="1.0" encoding=”iso-8859-9”?>
<!DOCTYPE OKUL SYSTEM " OKUL.DTD">
<SCHOOL>
  <NAME>&deu;</NAME>
  <FACULTY>
    <FID>ENGINERRING</FID>
    <DEPARTMENT>COMPUTER</DEPARTMENT>
    <PARTDEP>INDUSTRY</PARTDEP>
    ...
  </FACULTY>
  <FACULTY>
    <FID>LEARNIG</FID>
```

| Table 4.36 Continued... |
|---|
| `<DEPARTMENT>MATHMATIC</DEPARTMENT>`<br>`<PARTDEP>STATISTIC</PARTDEP>`<br>...<br>`</FACULTY>`<br>....<br>`</SCHOOL>` |
| **[SCHOOL.DTD]** |
| `<!ELEMENT SCHOOL (NAME, FACULTY+)>`<br>`<!ELEMENT NAME (#PCDATA)>`<br>`<!ELEMENT FACULTY (FID, PARTDEP+)>`<br>`<!ELEMENT FID (#PCDATA)>`<br>`<!ELEMENT PARTDEP (#PCDATA)>`<br>`<!ENTITY deu "Dokuz Eylül University">` |

In table 4.36, SCHOOL.DTD file defines root element and its descendants, its orders and its types and ENTITY definitions. DTD is similar to grammar declaration of XML document.

**Table 4.37 Document Type Definition Rule (WEB_12, 2004)**

| **Document Type Definition** | | |
|---|---|---|
| **[28] doctypedecl** | ::= | '<!DOCTYPE' S Name (S ExternalID)? S? ('[' (markupdecl \| PEReference \| S)* ']' S?)? '>'<br>[**VC**: Root Element Type] |
| **[29] markupdecl** | ::= | elementdecl \| AttlistDecl \| EntityDecl \| NotationDecl \| PI \| Comment<br>[**VC**: Proper Declaration/PE Nesting]<br>[**WFC**: PEs in Internal Subset] |

DTD declaration starts with *<!DOCTYPE* string, then a white-space character, a valid XML Name, an optional white-space character, an ExternalID and again optional white-space character, a [ character, a markupdecl element, a PEReference element or a white-space character and finally end with an ] character, optional white-space character and an > character. Markup declaration (markupdecl) can be in the six declaration form;

- (1)Element Declaration(elementdecl)
- (2)Attribute List Declaration (AttlistDecl)

- (3)Entity Declaration(EntityDecl)
- (4)Notation Declaration (NotationDecl)
- (5)Processing Instruction (PI)
- (6)Comments

**Table 4.38 XML Sample which has all declarations**

```
<?xml version="1.0" encoding="iso-8859-9"?>
 <!DOCTYPE SCHOOL [
 <!NOTATION EPS PUBLIC "+//ISBN 0-201-18127-4::Adobe//
 NOTATION PostScript Language Ref. Manual//EN">
 <!ENTITY file SYSTEM "image/image.eps" NDATA EPS>
 <!ENTITY % nameDecl "<!ELEMENT NAME (#PCDATA)>">
 <!ENTITY deu  "Dokuz Eylül University">
 <!ELEMENT NAME (NAME,IMAGE)>
  %nameDecl;
 <!ELEMENT IMAGE EMPTY>
 <!ATTLIST IMAGE PATH ENTITY #REQUIRED>
 <!-- DTD Ends -->
 ]>
<SCHOOL>
     <NAME>&deu;</NAME>
     <IMAGE PATH="file"/>
</SCHOOL>
```

In table 4.38, an XML document sample which has all XML declarations is shown. DTD and XML document is mixed. This type of DTD documents are named as internal DTD declaration. SCHOOL root element is made NAME and IMAGE elements. NAME element's declaration is defined in the *nameDecl* parameter entity. Parameter entities are XML units which provide to define content of markups. IMAGE element is an empty element and has no content. Content of PATH attribute is *file* entity which accepts *EPS* file format. EPS definition is a notation declaration.

Constraints in [28] and [29] EBNF rules;

- *"VC: Root Element Type"* (WEB_12, 2004); The Name in the document type declaration must match the element type of the root element. This validity constraint tells us that the Name in a document type declaration can't be just any Name. It has to be the element type name of the root element of the document.
- *"VC: Proper Declaration/PE Nesting;* Parameter-entity replacement text must be properly nested with markup declarations" (WEB_12, 2004).
- *"WFC: PEs in Internal Subset"* (WEB_12, 2004); In the internal DTD subset, parameter-entity references can occur only where markup declarations can occur, not within markup declarations.

**Table 4.39 WFC for internal DTD declarations**

```
<?xml version="1.0" encoding="iso-8859-9"?>
 <!DOCTYPE SCHOOL [
<!ELEMENT OKUL (NAME,CITY)>
<!--Since it hold the whole declaration of element, it is a valid parameter entity
definition -->
<!ENTITY % nameDecl "<!ELEMENT NAME (#PCDATA)>">
  %nameDecl;
<!--Since it hold the part of declaration of element, it is not a valid parameter entity
definition -->
    <!ENTITY % cityDecl "CITY (#PCDATA)">
    <!ELEMENT %cityDecl ;>
-->
<!ELEMENT CITY (#PCDATA)>
]>
<SCHOOL>
     <NAME>Dokuz Eylül University</NAME>
     <CITY>İzmir</CITY>
</SCHOOL>
```

In table 4.39, XML document's root element has the name SCHOOL which contains NAME and CITY element. NAME element is defined using parameter entity reference. *nameDecl* reference includes all information which will be used in element declaration. So it is also a valid declaration. But, PE reference that is defined for CITY element is not fulfilled, because it does not include the characters <, >,! and the ELEMENT keyword. Usage of *cityDecl* entity like <!ELEMENT %cityDecl;> does not fit WFC.

**Table 4.40 External Subset Rule (WEB_12, 2004)**

| **External Subset** | | |
|---|---|---|
| **[30] extSubset** | ::= | TextDecl? ExtSubsetDecl |
| **[31] extSubsetDecl** | ::= | ( markupdecl \| conditionalSect \| PEReference \| S )* |

External subset consists of an optional text declaration followed by an external subset declaration. Rule 31 shows that the latter is a combination of zero or more of the markup declarations, conditional sections, and parameter-entity references.

External subsets and external PE references has rules and declarations like internal ones. If an external subset is wanted use from an internal DTD, an URI (Uniform Resource Identifier) reference to external subset must be constructed. (URI) is a notation for naming resources on the Web. A URL (Uniform Resource Locator) such as *http://www.deu.edu.tr* is one kind of URI.

**Table 4.41 URI references**

| |
|---|
| <?xml version="1.0" encoding="iso-8859-9"?> <!DOCTYPE SCHOOL **SYSTEM "ext.DTD"** [ ... |
| <?xml version="1.0" encoding="iso-8859-9"?> <!DOCTYPE SCHOOL **SYSTEM "http://www.deu.edu.tr/ext.DTD"** [... |

Well-formed constraints for parameter entity references are not true for external parameter entities (WEB_12, 2004). It means that external parameter entities must not define all part of declaration.

**Table 4.42 Using PE references in external subsets**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<!DOCTYPE SCHOOL SYSTEM "ext.DTD" [
<!ELEMENT SCHOOL (NAME,CITY)>
<!--Since it hold the whole declaration of element, it is a valid parameter entity
```

**Table 4.42 Continued...**

```
definition -->
<!ENTITY % nameDecl "<!ELEMENT NAME (#PCDATA)>">
  %nameDecl;
]>
<SCHOOL>
     <NAME>Dokuz Eylül University</NAME>
     <CITY>İzmir</CITY>
</SCHOOL>
```

**EXT.DTD**

```
<!-- Although it does not hold the whole declaration of element, it is a valid
parameter
     entity definition, because  this DTD is used as external subset
-->

<!ENTITY % cityDecl "CITY (#PCDATA)">
<!ELEMENT %cityDecl;>
```

"If both the external and internal subsets are used, the internal subset is considered to occur before the external subset. This has the effect that entity and attribute-list declarations in the internal subset take precedence over those in the external subset" (WEB_12, 2004). In table 4.42, *schoolname* entity reference is used both in internal DTD and in external DTD. Since the priority of internal DTD, SCHOOL element will has *DEU* content.

**Table 4.43 Using internal and external declaration together**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<!DOCTYPE SCHOOL SYSTEM "ext.DTD" [
<!ELEMENT SCHOOL (#PCDATA)>
<!ENTITY schoolname "DEU">
]><SCHOOL>&schoolname;</SCHOOL>
```

**EXT.DTD**

```
<!ENTITY schoolname "Dokuz Eylul University">
```

## 4.12 Standalone Document Declaration

An XML document can get by with no declarations at all. It can also have declarations as part of an internal subset, and it can have declarations in an external subset such as a separate DTD file.

**Table 4.44 Standalone Document Declaration Rule (WEB_12, 2004)**

| Standalone Document Declaration | | |
|---|---|---|
| [32] SDDecl | ::= | S 'standalone' Eq (("'" ('yes' \| 'no') "'") \| ('"' ('yes' \| 'no')'"')) [VC: Standalone Document Declaration] |

According production rule, SDDecl element starts with white-space character, then follows with *standalone* keyword string, = sign character and *yes-no* values. "If the value *yes,* indicates that there are no markup declarations external to the document entity which affects the information passed from the XML processor to the application. The value *no* indicates that there are or may be such external markup declarations" (WEB_12, 2004). If there are no external markup declarations, the standalone document declaration has no meaning. If there are external markup declarations but there is no standalone document declaration, the value *no* is assumed.

**Table 4.45 Valid and non-valid Standalone attributes**

| Valid standalone attributes | Non-valid standalone attributes |
|---|---|
| Standalone="yes" | Standalone="yes' |
| Standalone='yes' | Standalone='yes" |
| Standalone="no" | Standalone="no' |
| Standalone='no' | Standalone='no" |

According to 32nd EBNF rule, the circumstances that value of standalone attribute can not be *no*;

- If default values of attributes is located outside of the document.
- If XML document have elements or attributes which have external values.

- If XML document has entity references except amp, lt, gt, apos and quot.

## 4.13 White-Spaces

Management of white spaces in XML document is a complex job like in all text based technologies. Especially, when XML documents are converted or transformed to other type of documents, impotency of white-space is increases. For example, two XML documents below may look like same with their contents but XML parser processes two documents differently.

**Table 4.46 White-spaces**

| | |
|---|---|
| <xml version="1.0"><br><ROOT><br><SubElement>1</SubElement><br><SubElement>2</SubElement><br></ROOT> | <xml version="1.0"><br><ROOT>   <SubElement> 1 </SubElement><br>    <SubElement>  2   </SubElement><br></ROOT> |

In XML documents, *xml:space* attribute is used to manage white-spaces. This attribute can take two values;

- *default;* XML parser manages white-space management itself. If xml:space attribute is not specified, "default" value is processed
- *preserve;* Provides to keep spaces.

**Table 4.47 Usage xml:space attribute**

| | |
|---|---|
| <PERSONNEL><br> <CODE>00001</CODE><br> <ADR xml:space="preserve"><br>      Kazım Dirik Mah. Gediz Cad.<br>      Apt. No 344  -   Kat 5<br>      BORNOVA<br> </ADR ><br></PERSONNEL> | <PERSONNEL xml:space="preserve"><br> <CODE>00001</CODE><br> <ADR>Kazım Dirik Mah. Gediz Cad.<br>      Apt. No 344  -   Kat 5<br>      BORNOVA<br> </ADR ><br></PERSONNEL> |

In table 4.47, two different usages of *preserve* attribute in the same XML document is showed. In the first one, it is assigned to ADR and in the second one it is assigned to root element PERSONNEL. Two XML documents "ADR" element's white-spaces are kept. But in the second XML document all elements' white-spaces are kept.

## 4.14 Language Identification

"In document processing, it is often useful to identify the natural or formal language in which the content is written. xml:lang attribute may be inserted in documents to specify the language used in the contents of any element in document" (WEB_12, 2004).

**Table 4.48 Language Identification Rule (WEB_12, 2004)**

| Language Identification | | |
|---|---|---|
| **[33 ] LanguageID** | ::= | Langcode ('-' Subcode)* |
| **[34 ] Langcode** | ::= | ISO639Code \|IanaCode \|UserCode |
| **[35 ] ISO639Code** | ::= | ([a-z ] \| [A-Z ]) ([a-z ] \| [A-Z ]) |
| **[36 ] IanaCode** | ::= | ('i ' \| 'I ') '-' ([a-z ] \| [A-Z ])+ |
| **[37 ] UserCode** | ::= | ('x ' \| 'X ')'-' ([a-z ] \| [A-Z ])+ |
| **[38 ] Subcode** | ::= | ([a-z ] \| [A-Z ])+ |

*LanguageID*, starts with the *Langcode* element. V alues that L angcode e lement can take are below;

A two-letter language code as defined by [ISO 639], Codes for the representation of names of languages. There exist 2704 language codes.

**Table 4.49 Valid and non-valid ISO639Code codes (WEB_12, 2004)**

| Valid ISO639Code codes | Non-valid ISO639Code codes |
|---|---|
| En | English |
| Tr | Turkish |
| TR | TURKISH |
| FR | French |
| Tr | Spanish |
| JP | Japan |
| tR | Turkey |

A language identifier registered with the IANA (Internet Assigned Numbers Authority). Begins with the -i or -I prefixes.

**Table 4.50 Valid and non-valid IanaCode codes (WEB_12, 2004)**

| Valid IanaCode codes | Non-valid IanaCode codes |
|---|---|
| i-no-bok | No-bok |
| i-no-nyn | No-nyn |
| i-navajo | Navajo |
| i-mingo | Mingo |

A language identifier assigned by the user, or agreed on between parties in private use; these must begin with the prefix x- or X- in order to ensure that they do not conflict with names later standardized or registered with IANA.

**Table 4.51 Valid and non-valid UserCode codes**

| Valid UserCode codes | Non-valid UserCode codes |
|---|---|
| x-klingon | Klingon |
| X-Elvish | Elvish |

In table 4.52, color keyword's usage in different grammars (United States and Great Britain) is showed.

**Table 4.52 xml:lang attribute**

```
<xml version="1.0">
<list>
  <p xml:lang="en-GB ">What colour is it?</p>
  <p xml:lang="en-US ">What color is it?</p>
</list>
```

## 4.15 Elements

"Each XML document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type which is defined in a related DTD or XML schema" (WEB_12, 2004). Each XML element can have one or more attributes.

**Table 4.53 Element Rule (WEB_12, 2004)**

| **Element** | |
|---|---|
| **[39 ] element ::=** | EmptyElemTag \| STag content Etag [**WFC:**Eleament Type Match ] [**VC:**Element Valid ] |

"**Element Type Match (WFC):** The Name in an element's end-tag must match the element type in the start-tag" (WEB_12, 2004).

"**Element Valid (VC):** An element is valid if there is a declaration matching elementdecl where the Name matches the element type, and one of the following holds" (WEB_12, 2004):

- If *EMPTY*, element must not include content
- If *ANY*, all children's types, sequences must be defined properly.
- If *Mixed*, the content of element must include both character data and child elements. All children's types, sequences must be defined properly.

**Table 4.54 Element Rule (2) (WEB_12, 2004)**

| **Element** |
|---|
| **[40] Stag** ::= '<' Name (S Attribute)*S?'>' [**WFC:**Unique Att Spec ] |
| **[41] Attribute ::=** Name Eq AttValue [**VC:**Attribute Value Type ] [**WFC:**No External Entity References ] [**WFC:**No < in Attribute Values ] |

Stag starts with < character, then it is followed 5th *Name* in the start-tags and end-tags gives the element's type. The Name-AttValue pairs are referred to as the attribute specifications of the element, with the Name in each pair referred to as the attribute name and the content of the AttValue as the attribute value.

**“Unique Att Spec (WFC):** No attribute name may appear more than once in the same start-tag or empty-element tag” (WEB_12, 2004).

**Table 4.55 Valid and non-valid start-elements**

| Valid start tags | Non-valid start tags |
|---|---|
| <CATEGORY> | < CATEGORY> |
| <CATEGORY > | <1CATEGORY> |
| <______> | < > |
| <CATEGORY ID=”0” NAME=”DIGITAL”> | |

Attribute starts with Name element followed by an “=” equal character which is followed attribute name.

**“Attribute Value Type (VC):** The value of an attribute must be of the type declared for it” (WEB_12, 2004).

**“No External Entity References (WFC):** Attribute values can not contain direct or indirect entity references to external entities” (WEB_12, 2004).

**“No < in Attribute Values (WFC):** The replacement text of any entity referred to directly or indirectly in an attribute value must not contain a <” (WEB_12, 2004).

**Table 4.56 Valid and non-valid attributes**

| Valid attributes | Non-valid attributes |
|---|---|
| name=“digital” | name=“digital’ |
| name=’digital’ | name=’digital” |
| Att1=“x &amp; y” | att1=“x & y” |
| Att1=“x &lt; y” | att1=“x < y” |
| att1=“1234” | att1=1234 |

**Table 4.57 Element Rule (3) (WEB_12, 2004)**

| Element |
|---|
| **[42] Etag** ::= ‘</’ Name S? ‘>’ |

Etag element starts with string </, then followed by an XML name, optionally followed by white-space, followed by the >character.

**Table 4.58 Valid and non-valid close-tags**

| Valid closed-tags | Non-valid closed tags |
|---|---|
| </CATEGORY> | </ CATEGORY> |
| </CATEGORY > | </1CATEGORY> |
| </______> | </> |
| </CATEGORYNAME> | </CATEGORY NAME=”digital”> |

**Table 4.59 Element Rule (4) (WEB_12, 2004)**

| Element |
|---|
| **[43] content ::=** (element \| CharData \| Reference \| CDSect \| PI \| Comment)* |

Content element can be form of any number of elements, character data, references, CDATA sections, processing instructions, and comments in any order. This production lists everything that can appear inside an element.

**Table 4.60 Element Rule (5) (WEB_12, 2004)**

| Element |
|---|
| **[44] EmptyElemTag ::=** ‘<’ Name (S Attribute)* S? ‘/>’ |

Empty-element tags may be used for any element which has no content, whether or not it is declared using the keyword EMPTY. EmptyElemTag starts with the character <, followed by an XML name, followed by white-space, followed by zero more attributes separated from each other by white-space, optionally followed by white-space, followed by the string “/>”. For example <img> and <br> elements are empty elements in HTML.

**Table 4.61 Valid and non-valid empty elements**

| Valid empty elements | Non-valid empty elements |
|---|---|
| <CATEGORY/> | < CATEGORY/> |
| </CATEGORY/ > | <1CATEGORY/> |
| </______> | </> |
| <CATEGORY NAME=”digital” /> | </CATEGORY NAME=”digital”> |
| </CATEGORY NAME=”digital” /> | </CATEGORY/> |

## 4.16 Element Type Declarations

Element type declarations are used for to give some constraint to element content. Element type declarations often constraints the element types that can appear as children (WEB_12, 2004).

**Table 4.62 Element Type Declaration Rule (WEB_12, 2004)**

| **Element Type Declaration** |
|---|
| **[45] elementdecl** ::= '<!ELEMENT' S Name S contentspec S? '>' [ **VC:**Unique Element Type Declaration ] |
| **[46] contentspec** ::= 'EMPTY' \| 'ANY' \| Mixed \| children |

Elementdecl element starts with string <!ELEMENT, followed by white-space, followed by an XML name ([5]), followed by a content specification ([46]), optionally followed by white-space, followed by the >character.

**"Unique Element Type Declaration (VC) :** No element type may be declared more than once" (WEB_12, 2004).

**Table 3.63 Element type definitions**

| |
|---|
| <!ELEMENT img EMPTY> |
| <!ELEMENT temp ANY> |
| <!ELEMENT temp1 (#PCDATA \| temp2)*> |

## 4.17 Element Content

"An element type has element content when elements of that type must contain only child elements, optionally separated by white space. In this case, the constraint includes a content model, a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear" (WEB_12, 2004).

**Table 4.64 Element Content Models Rule (WEB_12, 2004)**

| **Element Content Models** | | |
|---|---|---|
| **[47] children** | ::= | (choice \| seq) ('?' \| '*' \| '+')? |
| **[48] cp** | ::= | (Name \| choice \| seq) ('?' \| '*' \| '+')? |
| **[49] choice** | ::= | '(' S? cp ( S? '\|' S? cp )* S? ')' [**VC**:Proper Group/PE Nesting] |
| **[50] seq** | ::= | '(' S? cp ( S? ',' S? cp )* S? ')' [**VC**:Proper Group/PE Nesting] |

Children element can be form of choice or seq element in any order or ?, * and + characters zero or one times. Cp (content particle) can be form as an XML name, choice, or sequence, optionally suffixed with a ?, *, or +. Choice element may have one or more content particles enclosed in parentheses and separated from each other by vertical bars and optional white-space.

**Table 4.65 Choice usage**

| |
|---|
| (EL1 \| EL2 \| EL3 \| EL4 \| EL5 \| EL6 \| EL7 \| EL8? \| EL9) |
| (EL1\|EL2\|EL3\|EL4\|EL5\|EL6\|EL7\|EL8?\|EL9) |
| (MAN \| WOMAN) |
| ( MAN \| WOMAN ) |
| ( FAMILY \| (MOM, DAD, CHILD+) ) |

Seq element (sequence) may have one or more content particles ([48]) enclosed in parentheses and separated from each other by commas and optional whites-pace.

**Table 4.66 Seq usage**

| |
|---|
| (EL1 , EL2 , EL3 , EL4 , EL5 , EL6 , EL7 , EL8? , EL9) |
| (EL1,EL2,EL3,EL4,EL5,EL6,EL7,EL8?,EL9) |
| (MAN,WOMAN) |
| ( MAN , WOMAN ) |
| ( FAMILY ) |
| ( MEMBER , (MOM \| DAD\| CHILD?) ) |

**"Proper Group/PE Nesting (VC):** Parameter-entity replacement text must be properly nested with parenthesized groups. If either of the opening or closing parentheses in a choice , seq , or Mixed construct is contained in the replacement text for a parameter entity, both must be contained in the same r eplacement text" (WEB_12, 2004).

## 4.18 Mixed Content

If an element type includes both character data and child elements, it is called element with mixed content.

**Table 4.67 Mixed Content Declaration Rule (WEB_12, 2004)**

| Mixed-Content Declaration |
|---|
| **[51] Mixed** ::= '(' S? '#PCDATA' (S? '\|' S? Name)* S? ')*'<br>\| '(' S? '#PCDATA' S? ')' |

Mixed element (mixed content) is either the string (#PCDATA) or a choice that includes the string #PCDATA as its first content particle.

**Table 4.68 Valid and non-valid mixed contents**

| Valid mixed contents | Non-valid mixed contents |
|---|---|
| (#PCDATA) | () |
| ( #PCDATA ) | (CATEGORY \| #PCDATA) |
| (#PCDATA \| CATEGORY) | (#PCDATA \| EL1 \| #PCDATA \| EL3) |
| ( #PCDATA \| CATEGORY ) | ( #PCDATA \| (CATEGORY,EL1,EL2) ) |
| (#PCDATA \| EL1 \| EL2 \| EL3) | |

## 4.19 Attribute-List Declarations

Attributes are special XML units which are name-value pairs. Attribute specifications may appear only within start-tags and empty-element tags. Usage of attributes may be as below;

- To bring some properties to elements
- Giving constraints to element attributes

- Giving default values to element attributes

**Table 4.69 Attribute-List Declaration Rule (WEB_12, 2004)**

| **Attribute-List Declaration** | | |
|---|---|---|
| **[52] AttlistDecl** | ::= | '<!ATTLIST' S Name AttDef* S? '>' |
| **[53] AttDef** | ::= | S Name S AttType S DefaultDecl |

AttlistDecl element consists of the keyword string <!ATTLIST , followed by white-space, followed by an XML name, followed by zero or more attribute definitions, optionally followed by white-space, followed by the > character.

**Table 4.70 Attribute-list declarations**

| |
|---|
| <!ATTLIST CATEGORY NAME CDATA #REQUIRED> |
| <!ATTLIST CATEGORY NAME CDATA #IMPLIED> |
| <!ATTLIST CATEGORY NAME CDATA #FIXED "DIJITAL"> |
| <!ATTLIST CATEGORY NAME NMTOKEN #REQUIRED> |
| <!ATTLIST CATEGORY NAME NMTOKENS #REQUIRED> |
| <!ATTLIST TEL NO ID #REQUIRED> |
| <!ATTLIST TELNOS PLACE ENTITIES #REQUIRED> |

AttDef element ( attribute definition) starts with white-space, an XML name, more white-space, an attribute type, more white-space, and a default declaration.

**Table 4.71 Attribute declarations**

| |
|---|
| CATEGORY NAME CDATA #REQUIRED |
| CATEGORY NAME CDATA #IMPLIED |
| CATEGORY NAME CDATA #FIXED "DIJITAL" |
| TEL NO ID #REQUIRED |
| TELNOS YER ENTITIES #REQUIRED |

## 4.20 Attribute Types

Attribute types can be examined in three categories, string type, tokenized type and enumerated type.

**Table 4.72 Attribute Types Rule (WEB_12, 2004)**

| **Attribute Types** | | | |
|---|---|---|---|
| **[54] AttType** | ::= | StringType \| TokenizedType \| EnumeratedType | |
| **[55] StringType** | ::= | 'CDATA' | |
| **[56] TokenizedType** | ::= | 'ID' | [VC:ID] |
| | | | [VC:One ID per ElementType] |
| | | | [VC:ID Attribute Default] |
| | | \| 'IDREF' | [VC:IDREF] |
| | | \| 'IDREFS' | [VC:IDREF] |
| | | \| 'ENTITY' | [VC:Entity Name] |
| | | \| 'ENTITIES' | [VC:Entity Name] |
| | | \| 'NMTOKEN' | [VC:Name Token] |
| | | \| 'NMTOKENS' | [VC:Name Token] |

AttType element can be in the form of StringType, TokenizedType or EnumeratedType. StringType element specifies strings as CDATA type. TokenizedType element can be form in the seven strings ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN or NMTOKENS. Validation constraints for TokenizedType element are below;

- "*VC:ID*, Values of type ID must match the Name production. A name must not appear more than once in an XML document as a value of this type" (WEB_12, 2004).
- "*VC:One ID per ElementType*, No element type may have more than one ID attribute specified" (WEB_12, 2004).
- "*VC:ID Attribute Default*, An ID attribute must have a declared default of #IMPLIED or #REQUIRED" (WEB_12, 2004).
- "*VC:IDREF*, Values of type IDREF must match the Name production, and values of type IDREFS must match Names" (WEB_12, 2004).
- "*VC:Entity Name,* Values of type ENTITY must match the Name production, values of type ENTITIES must match Names, each Name must match the name of an unparsed entity declared in the DTD" (WEB_12, 2004).

- "*VC:Name Token*, Values of type NMTOKEN must match the Nmtoken production; values of type NMTOKENS must match Nmtokens" (WEB_12, 2004).

## 4.21 Enumerated Attributes

**Table 4.73 Enumerated Attributes (WEB_12, 2004)**

| Enumerated Attributes |
|---|
| **[57] EnumeratedType** ::= NotationType \| Enumeration |
| **[58] NotationType** ::= 'NOTATION' S '(' S? Name (S? '\|' S? Name)* S? ')' [VC:Notation Attributes] |
| **[59] Enumeration** ::= '(' S? Nmtoken (S? '\|' S? Nmtoken)* S? ')' [VC:Enumeration] |

EnumeratedType element can be either a notation type or an enumeration. NotationType element starts with the string keyword NOTATION, followed by white-space, followed by one or more XML names, separated by vertical bars, and enclosed in parentheses.

**Table 4.74 Valid and non-valid notation definitions**

| Valid notation definitions | Non-valid notation definitions |
|---|---|
| NOTATION (BCC) | NOTATION ("BCC") |
| NOTATION (BCC \| PDF) | NOTATION (BCC PDF) |
| NOTATION (bcc \| gcc \| delphi32) | NOTATION (bcc, gcc, delphi32) |
| NOTATION (A \| B \| C) | NOTATION ("A" "B" "C") |

Enumeration element as one or more XML name tokens separated by vertical bars and enclosed in parentheses.

"**Enumeration (VC):** Values of this type must match one of the Nmtoken tokens in the declaration" (WEB_12, 2004).

**Table 4.75 Valid and non-valid enumeration types**

| Valid enumeration types | Non-valid enumeration types |
|---|---|
| (mom) | () |
| (mom \| dad \| brother) | (mom dad brother) |
| ( mom \| dad \| brother ) | ( mom , dad , brother) |
| ( AT1 \| AT2 \| AT3 \| AT4 \| AT5 ) | AT1 \| AT2 \| AT3 \| AT4 \| AT5 |

## 4.22 Attribute Defaults

Attribute declarations provide information about attribute's presence. Three attribute defaults exist.

**Table 4.76 Attribute Defaults Rule (WEB_12, 2004)**

| Attribute Defaults |
|---|
| **[60] DefaultDecl** ::= '#REQUIRED' \| '#IMPLIED' \| (('#FIXED' S)? AttValue) [**VC:**Required Attribute][**VC:**Attribute Default Legal] [**WFC:**No <in Attribute Values][**VC:**Fixed Attribute Default] |

- #REQUIRED, defined element attribute must be included within the XML document.
- #IMPLIED, defined element attribute can not have default attribute value.
- #FIXED, defined element attribute must always have the default attribute value.

DefaultDecl element has three validity constraints;

- **"Required Attribute:** If the default declaration is the keyword string #REQUIRED, then the attribute must be specified for all elements of the type in the attribute-list declaration" (WEB_12, 2004).
- **"Attribute Default Legal:** The default value must meet the constraints of the declared attribute type" (WEB_12, 2004).
- **"Fixed Attribute Default:** If an attribute has a default value declared with the #FIXED keyword, instances of that attribute must match the default value" (WEB_12, 2004).

**Table 4.77 Attribute-list declarations**

| |
|---|
| `<!ATTLIST Personnel id ID #REQUIRED name CDATA #IMPLIED>` |
| `<!ATTLIST List tipi (ordered \| unordered) “ordered”>` |
| `<!ATTLIST Country code CDATA #FIXED “TR”>` |

## 4.23 Attribute-Value Normalization

XML processor's normalization steps before the attributes values are controlled or validated:

- A character reference is processed by appending the referenced character to the attribute value.
- An entity reference is processed by recursively processing the replacement text of the entity.
- A white-space character (#x20, #xD, #xA, #x9) is processed by appending #x20 to the normalized value, except that only a single #x20 is appended for a "#xD#xA" sequence that is part of an external parsed entity or the literal entity value of an internal parsed entity.
- Other characters are processed by appending them to the normalized value.

If the declared value is not CDATA, then the XML processor must further process the normalized attribute value by discarding any leading and trailing space (#x20) characters, and by replacing sequences of space (#x20) characters by a single space (#x20) character.

## 4.24 Conditional Sections

“Conditional sections are units in document type declaration external subset which are included in, or excluded from, the logical structure of the DTD based on the keyword” (Harold E.R, 1999). Conditional sections can include comment lines, processing instructions, element declarations and other conditional sections.

If the keyword in conditional section is INCLUDE, the part within the keyword is accepted as the DTD part. If the keyword in conditional section is IGNORE, the part within the keyword is not accepted as the DTD part. An INCLUDE part in the IGNORE part is discarded (WEB_12, 2004).

**Table 4.78 Conditional Sections Rule (WEB_12, 2004)**

| **Conditional Sections** |
|---|
| **[61] conditionalSect** ::= includeSect \| ignoreSect |
| **[62] includeSect** ::= ‘<![‘ S? ‘INCLUDE’ S? ‘[‘ extSubsetDecl ‘]]>’ |
| **[63] ignoreSect** ::= ‘<![‘ S? ‘IGNORE’ S? ‘[‘ iganoreSectContents* ‘]]>’ |
| **[64] ignoreSectContents** ::= Ignore (‘<![‘ ignoreSectContents ‘]]>’ Ignore)* |
| **[65] Ignore** ::= Char* - (Char* (‘<![‘ \| ‘]]>’) Char*) |

ConditionalSect element can be either an include section or an ignore section. IncludeSect element (include section) can be an external subset declaration between <![INCLUDE [ ]]>, modulo white-space.

**Table 4.79 INCLUDE sections**

| |
|---|
| <![ INCLUDE [ ]]> |
| <![INCLUDE [ ]]> |
| <![ INCLUDE [ ]]> |

IgnoreSect element defines an ignore section as ignore section contents ([64]) between <![IGNORE [ ]]>, modulo white-space. IgnoreSectContents element defines an ignore section contents as an ignore block ([65]), optionally followed by a block of text sandwiched between <![ and ]]> strings, followed by more text. Ignore element defines an ignore block as any run of text that contains neither the <![or ]]>literals.

**Table 4.80 IGNORE sections**

| |
|---|
| <![ IGNORE [ ]]> |
| <![IGNORE [ ]]> |
| <![ IGNORE [ ]]> |

**Table 4.81 Conditional sections**

```
<![INCLUDE[
  <!ELEMENT book (comment*,title, content, add?)>
]]>
<![IGNORE[
  <!ELEMENT book (title, content, add?)>
]]>
```

## 4.25 Character And Entity References

Character references are referred to characters in the ISO/IEC 1046 character-set and range specified in 2nd Char rule.

**Table 4.82 Character Reference Rule (WEB_12, 2004)**

| **Character Reference** | | |
|---|---|---|
| **[66] CharRef** | ::= | '&#' [0-9]+ ';' \| '&#x' [0-9a-fA-F]+ ';' [**WFC:**Legal Character] |

CharRef element defines two forms for character references. The first is the string &# followed by one or more of the ASCII digits 0 through 9. The second form is the string &#x followed by one or more of the hexadecimal digits 0 through F. The digits representing 10 through 16 may be either lower- or uppercase.

"**Legal Character (WFC):** Characters referred to using character references must match the production for Char" (WEB_12, 2004).

Entities can be either parsed or unparsed;

- Parsed entities' contents are atomic unit within the XML document.
- Unparsed entities are references to external sources. Source may not be simple text. Each unparsed entity must have a related notation type declaration to define external resource's format.

**Table 4.83 Entity Reference Rule (WEB_12, 2004)**

| **Entity Reference** | | |
|---|---|---|
| **[67] Reference** | ::= | EntityRef \| CharRef |
| **[68] EntityRef** | ::= | '&' Name ';'<br>[**WFC:**Entity Declared]<br>[**VC:**Entity Declared]<br>[**WFC:**Parsed Entity]<br>[**WFC:**No Recursion] |
| **[69] PEReference** | ::= | '%' Name ';'<br>[**VC:**Entity Declared]<br>[**WFC:**No Recursion]<br>[**WFC:**In DTD] |

Reference element defines a reference as either an entity reference or a character reference. EntityRef element defines an entity reference as an XML name between the ampersand character and a semicolon. PEReference element defines a parameter entity reference as an XML name between the percent character and a semicolon.

Well-formed constraints and validity constraints fro references are below;

- **"Entity Declared (WFC):** In a document without any DTD, a document with only an internal DTD subset which contains no parameter entity references, or a document with "standalone='yes'", the Name given in the entity reference must match that in an entity declaration" (WEB_12, 2004).
- **"Entity Declared (VC):** In a document with an external subset or external parameter entities with "standalone='no'", the Name given in the entity reference must match that in an entity declaration. Valid documents should declare the entities amp, lt, gt, apos, quot" (WEB_12, 2004).

- **"Parsed Entity (WFC):** An entity reference must not contain the name of an unparsed entity. Unparsed entities may be referred to only in attribute values declared to be of type ENTITY or ENTITIES" (WEB_12, 2004).
- **"No Recursion (WFC):** A parsed entity must not contain a recursive reference to itself, either directly or indirectly" (WEB_12, 2004).
- **"In DTD (WFC):** Parameter-entity references may only appear in the DTD" (WEB_12, 2004).

**Table 4.84 Valid and non-valid entity references**

| Valid entity references | Non-valid entity references |
|---|---|
| &lt; | &lt |
| &gt; | & gt; |
| &apos; | & apos ; |

**Table 4.85 Valid and non-valid parameter entity references**

| Valid parameter entity references | Non-valid parameter entity references |
|---|---|
| %name; | %name |
| %per_no; | & per_no; |
| %per_no; | %per no; |

## 4.26 Entity Declarations

If entity is declared like in the rule EntityValue, it is called as internal entity the other entities are called as external entities.

**Table 4.86 Entity Declarations Rule (WEB_12, 2004)**

| Entity Declarations | | |
|---|---|---|
| **[70] EntityDecl** | ::= | GEDecl \| PEDecl |
| **[71] GEDecl** | ::= | '<!ENTITY' S Name S EntityDef S? '>' |
| **[72] PEDecl** | ::= | '<!ENTITY' S '%' S Name S PEDef S? '>' |
| **[73] EntityDef** | ::= | EntityValue \| (ExternalID NDataDecl?) |
| **[74] PEDeaf** | ::= | EntityValue \| ExternalID |

EntityDecl element defines an entity declaration as either a general entity declaration or a parameter entity declaration. GEDecl element defines a general entity declaration as the literal <!ENTITY followed by white-space, followed by an XML name, followed by an entity definition, optionally followed by white-space, followed by the >character.

**Table 4.87 Valid and non-valid general entity declarations (WEB_12, 2004)**

| Valid general entity declarations | Non-valid general entity declarations |
|---|---|
| <!ENTITY DEU "Dokuz Eylul University"> | <!ENTITY DEU Dokuz Eylul University> |
| <!ENTITY img SYSTEM "img.gif"> | <!ENTITY img SYSTEM img.gif> |
| <!ENTITY copy "Copyright Netsis"> | <!ENTITY copy right "Copyright Netsis"> |

PEDecl element defines a parameter entity declaration as the string <!ENTITY followed by white-space, followed by a percent sign and more white-space, followed by an XML name, followed by an entity definition, optionally followed by white-space, followed by the ">" character.

**Table 4.88 Valid and non-valid parameter entity declarations**

| Valid parameter entity declarations | Non-valid parameter entity declarations |
|---|---|
| <!ENTITY % nameDecl "<!ELEMENT NAME (#PCDATA)>"> | <!ENTITY & nameDecl "<!ELEMENT NAME (#PCDATA)>"> |
| <!ENTITY % deu "Dokuz Eylul University"> | <!ENTITY % deu ; "Dokuz Eylul University"> |
| <!ENTITY % homeadr "(city \| town \| post \| apt)"> | <!ENTITY % homeadr "(city \| town \| post \| apt)"> |

**Table 4.89 External Entities Rule (WEB_12, 2004)**

| External Entities | | |
|---|---|---|
| **[75] ExternalID** | ::= | 'SYSTEM' S SystemLiteral<br>\| 'PUBLIC' S PubidLiteral S SystemLiteral |
| **[76] NDataDecl** | ::= | S 'NDATA' S Name<br>[**VC:**Notation Declared] |

ExternalID element is specialized for system or general. This element defines an external ID as either the keyword SYSTEM followed by white-space and a system literal or the keyword PUBLIC followed by white-space, a public ID literal, more white-space, and a system-literal.

**“Notation Declared (VC):** The Name must match the declared name of a notation” (WEB_12, 2004).

**Table 4.90 Valid and non-valid external entity declarations**

| Valid external entity declarations | Non-valid external entity declarations |
|---|---|
| SYSTEM “image.gif” | SYSTEM image.gif |
| SYSTEM “/images/image.gif” | SYSTEM “/images/image.gif ’ |
| SYSTEM “http://www.netsis.com.tr/image.gif” | SYSTEM http://www.netsis.com.tr/image.gif |
| SYSTEM “../images/image.gif” | SYSTEM ../images/image.gif |
| <!ENTITY file SYSTEM "image/file.eps" NDATA EPS> | <!ENTITY file SYSTEM image/file.eps NDATA EPS> |
| PUBLIC “-//IETF//NONSGML Media Type image/gif//EN” “http://www.isi.edu/in-notes/iana/assignments/media-types/image/gif” | PUBLIC “-//IETF//NONSGML Media Type image/gif//EN” |
| | PUBLIC “http://www.isi.edu/in-notes/iana/assignments/media-types/image/gif” |

## 4.27 Text Declaration

**Table 4.91 Text Declaration Rule (WEB_12, 2004)**

| Text Declaration | | |
|---|---|---|
| **[77] TextDecl** | ::= | ‘<?xml’ VersionInfo? EncodingDecl S? ‘?>’ |

TextDecl element defines text declaration almost like an XML declaration except that it may not have a standalone document declaration. It starts with string “<?xml”, followed by optional *VersionInfo* element followed by *EncodingDecl* element optional white-space character and “?>” string.

**Table 4.92 Valid and non-valid text declarations**

| Valid text declarations | Non-valid text declarations |
|---|---|
| <?xml version="1.0"?> | <?xml ?> |
| <?xml version="1.0" encoding="utf-8"?> | <?xml standalone="no"?> |
| | <?xml version="1.0" encoding="utf-8" standalone="no"?> |
| | <?xml encoding="utf-8" version="1.0"?> |

**Table 4.93 Well-formed External Parsed Entities Rule (WEB_12, 2004)**

| Well-formed External Parsed Entities | | |
|---|---|---|
| **[78] extParsedEnt** | ::= | TextDecl? Content |
| **[79] extPE** | ::= | TextDecl? extSubsetDecl |

extParsedEnt element consists of an optional text declaration followed by content. This content may not include a DTD or any markup declarations. ExtPE element consists of an optional text declaration followed by an external subset declaration.

## 4.28 Encoding Declarations

"Each external parsed entity in an XML document may use a different encoding for its characters. All XML processors must be able to read entities in either UTF-8 or UTF-16. Entities encoded in UTF-16 must begin with the *Byte Order Mark* described by ISO/IEC 10646 Annex E and Unicode Appendix B" (WEB_12, 2004).

**Table 4.94 Encoding Declarations Rule (WEB_12, 2004)**

| Encoding Declarations | | |
|---|---|---|
| **[80] EncodingDecl** | ::= | S 'encoding' Eq ('"' EncName '"' \| "'" EncName "'" ) |
| **[81] EncName** | ::= | [A-Za-z] ([A-Za-z0-9._] \| '-')* |

EncodingDecl element defines an encoding declaration as white-space followed by the string "" followed by an equals sign, followed by the name of the encoding enclosed in either single or double quotes. EncName element defines the name of an encoding begins with one of the ASCII letters *A* through *Z* or *a* through z, followed by any number of ASCII letters, digits, periods, underscores, and hyphens.

**Table 4.95 Valid and non-valid encoding declarations**

| Valid encoding declarations | Non-valid encoding declarations |
|---|---|
| encoding="utf-8" | encoding="utf-8' |
| encoding="iso-8859-9" | encoding=iso-8859-9 |
| encoding = "utf-8" | encoding = 'utf-8" |
| encoding = 'iso-8859-9' | encoding = 'iso-8859-9 |
| encoding = 'utf-16' | encoding = 'utf 16' |

## 4.29 Notation Declarations

Notations identify by name the format of unparsed entities, the format of elements.

**Table 4.96 Notation Declarations Rule (WEB_12, 2004)**

| Notation Declarations | | |
|---|---|---|
| **[82] NotationDecl** | ::= | '<!NOTATION' S Name S (ExternalID \| PublicID) S? '>' |
| **[83] PublicID** | ::= | 'PUBLIC' S PubidLiteral |

NotationDecl element starts with the string <!NOTATION, followed by white-space, followed by an XML name for the notation, followed by white-space, followed by either an external ID or a public ID, optionally followed by white-space, followed by the literal string "". Publilc element defines a public ID as the literal string PUBLIC, followed by white-space, and followed by a public ID literal.

**Table 4.97 Valid and non-valid notation declarations**

| Valid notation declarations | Non-valid notation declarations |
|---|---|
| <!NOTATION GIF SYSTEM "image/gif"> | <! NOTATION GIF SYSTEM "image/gif"> |
| <!NOTATION GIF SYSTEM "image/gif" > | < !NOTATION GIF SYSTEM "image/gif"> |
| <!NOTATION GIF PUBLIC "-//IETF//NONSGML Media Type image/gif//EN " | <! NOTATION GIF SYSTEM image/gif> |

| | |
|---|---|
| http://www.isi.edu/in-notes/iana/assignments/media-types/image/gif"> | <!NOTATION GIF "image/gif"> |

## 4.30 Character Classes

Three types of character classes exist;

- *BaseChar*

  Class includes alphabetic Unicode characters and does not contain digits and punctuation marks. Base character class includes Arabic, Hebrew, Cyril and other alphabets in addition to English alphabet.

- *Ideographic Characters*

  These characters are Chinese, Japanese and Korean alphabets which are located in the range of #x4E00-#x9FA5.

- *CombiningChar*

  These characters are the combination form of more than one base character. For example, "ğ" character is created using base character "g" and character "˘".

**Table 4.98 Characters Rule (WEB_12, 2004)**

| Characters |
|---|
| **[84] Letter** ::= BaseChar \| Ideographic |
| **[85] BaseChar** ::= [#x0041-#x005A] \| [#x0061-#x007A] \| [#x00C0-#x00D6] \| [#x00D8-#x00F6] \| [#x00F8-#x00FF] \| [#x0100-#x0131] \| [#x0134-#x013E] \| [#x0141-#x0148] \| [#x014A-#x017E] \| [#x0180-#x01C3] \| [#x01CD-#x01F0] \| [#x01F4-#x01F5] \| [#x01FA-#x0217] \| [#x0250-#x02A8] \| [#x02BB-#x02C1] \| #x0386 \| [#x0388-#x038A] \| #x038C \| [#x038E-#x03A1] \| [#x03A3-#x03CE] \| [#x03D0-#x03D6] \| #x03DA \| #x03DC \| #x03DE \| #x03E0 \| [#x03E2-#x03F3] \| [#x0401-#x040C] \| [#x040E-#x044F] \| [#x0451-#x045C] \| [#x045E-#x0481] \| [#x0490-#x04C4] \| [#x04C7-#x04C8] \| [#x04CB-#x04CC] \| [#x04D0-#x04EB] \| [#x04EE-#x04F5] \| [#x04F8-#x04F9] \| [#x0531-#x0556] \| #x0559 \| [#x0561-#x0586] \| [#x05D0-#x05EA] \| [#x05F0-#x05F2] \| [#x0621-#x063A] \| [#x0641-#x064A] \| [#x0671-#x06B7] \| [#x06BA-#x06BE] \| [#x06C0-#x06CE] \| [#x06D0-#x06D3] \| #x06D5 \| [#x06E5-#x06E6] \| [#x0905-#x0939] \| #x093D \| [#x0958-#x0961] \| [#x0985-#x098C] \| [#x098F-#x0990] \| [#x0993-#x09A8] \| |

**Table 4.98 Continued…**
[#x09AA-#x09B0] | #x09B2 | [#x09B6-#x09B9] | [#x09DC-#x09DD] |
[#x09DF-#x09E1] | [#x09F0-#x09F1] | [#x0A05-#x0A0A] | [#x0A0F-#x0A10] |
[#x0A13-#x0A28] | [#x0A2A-#x0A30] | [#x0A32-#x0A33] | [#x0A35-#x0A36] |
[#x0A38-#x0A39] | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74] |
[#x0A85-#x0A8B] | #x0A8D | [#x0A8F-#x0A91] | [#x0A93-#x0AA8] |
[#x0AAA-#x0AB0] | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0 |
[#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28] | [#x0B2A-#x0B30] |
[#x0B32-#x0B33] | [#x0B36-#x0B39] | #x0B3D | [#x0B5C-#x0B5D] |
[#x0B5F-#x0B61] | [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-#x0B95] |
[#x0B99-#x0B9A] | #x0B9C | [#x0B9E-#x0B9F] | [#x0BA3-#x0BA4] |
[#x0C0E-#x0C10] | [#x0C12-#x0C28] | [#x0C2A-#x0C33] | [#x0C35-#x0C39] |
[#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90] | [#x0C92-#x0CA8] |
[#x0CAA-#x0CB3] | [#x0CB5-#x0CB9] | #x0CDE | [#x0CE0-#x0CE1] |
[#x0D05-#x0D0C] | [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39] |
[#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33] |
[#x0E40-#x0E45] | [#x0E81-#x0E82] | #x0E84 | [#x0E87-#x0E88] | #x0E8A |
#x0E8D | [#x0E94-#x0E97] | [#x0E99-#x0E9F] | [#x0EA1-#x0EA3] | #x0EA5 |
#x0EA7 | [#x0EAA-#x0EAB] | [#x0EAD-#x0EAE] | #x0EB0 | [#x0EB2-#x0EB3] |
#x0EBD | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-#x0F69] |
[#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100 | [#x1102- #x1103] | [#x1105-
#x1107] | #x1109 | [#x110B-#x110C] | [#x110E-#x1112] | #x113C | #x113E | #x1140
| #x114C | #x114E | #x1150 | [#x1154-#x1155] | #x1159 | [#x115F-#x1161] | #x1163
| #x1165 | #x1167 | #x1169 | [#x116D-#x116E] | [#x1172-#x1173] | #x1175 | #x119E
| #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8] | #x11BA | [#x11BC-
#x11C2] | #x11EB | #x11F0 | #x11F9 | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] |
[#x1F00-#x1F15] | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D] |
[#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D | [#x1F5F-#x1F7D] |
[#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE | [#x1FC2-#x1FC4] |
[#x1FC6-#x1FCC] | [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC] |
[#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126 | [#x212A-#x212B] | #x212E |
[#x2180-#x2182] | [#x3041-#x3094] | [#x30A1-#x30FA] | [#x3105-#x312C] |
[#xAC00-#xD7A3]

**[86] Ideographic** ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]

**[87] CombiningChar ::=** [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486] |
[#x0591-#x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD]
| #x05BF | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652] | #x0670 |
[#x06D6-#x06DC] | [#x06DD-#x06DF] | [#x06E0-#x06E4] | [#x06E7-#x06E8] |
[#x06EA-#x06ED] | [#x0901-#x0903] | #x093C | [#x093E-#x094C] | #x094D
|0020[#x0951-#x0954] | [#x0962-#x0963] | [#x0981-#x0983] | #x09BC | #x09BE |
#x09BF | [#x09C0-#x09C4] | [#x09C7-#x09C8] | [#x09CB-#x09CD] | #x09D7 |
[#x09E2-#x09E3] | #x0A02 | #x0A3C | #x0A3E | #x0A3F | [#x0A40-#x0A42] |
[#x0A47-#x0A48] | [#x0A4B-#x0A4D] | [#x0A70-#x0A71] | [#x0A81-#x0A83] |
#x0ABC | [#x0ABE-# x0AC5] | [#x0AC7-#x0AC9] | [#x0ACB-#x0ACD] |

| **Table 4.98 Continued…** |
|---|
| [#x0B01-#x0B03] \| #x0B3C \| [#x0B3E-#x0B43] \| [#x0B47-#x0B48] \| [#x0B4B-#x0B4D] \| [#x0B56-#x0B57] \| [#x0B82-#x0B83] \| [#x0BBE-#x0BC2] \| [#x0BC6-#x0BC8] \| [#x0BCA-#x0BCD] \| #x0BD7 \| [#x0C01-#x0C03] \| [#x0C3E-#x0C44] \| [#x0C46-#x0C48] \| [#x0C4A-#x0C4D] \| [#x0C55-#x0C56] \| [#x0CD5-#x0CD6] \| [#x0D02-#x0D03] \| [#x0D3E-#x0D43] \| [#x0D46-#x0D48] \| [#x0D4A-#x0D4D] \| #x0D57 \| #x0E31 \| [#x0E34-#x0E3A] \| [#x0E47-#x0E4E] \| #x0EB1 \| [#x0EB4-#x0EB9] \| [#x0EBB-#x0EBC] \| [#x0EC8-#x0ECD] \| [#x0F18-#x0F19] \| #x0F35 \| #x0F37 \| #x0F39 \| #x0F3E \| #x0F3F \| [#x0F71-#x0F84] \| [#x0F86-#x0F8B] \| [#x0F90-#x0F95] \| #x0F97 \| [#x0F99-#x0FAD] \| [#x0FB1-#x0FB7] \| #x0FB9 \| [#x20D0-#x20DC] \| #x20E1 \| [#x302A-#x302F] \| #x3099 \| #x309A |

**Table 4.99 Characters Rule (2) (WEB_12, 2004)**

| **Characters** |
|---|
| **[88] Digit ::=** [#x0030-#x0039] \| [#x0660-#x0669] \| [#x06F0-#x06F9] \| [#x0966-#x096F] \| [#x09E6-#x09EF] \| [#x0A66-#x0A6F] \| [#x0AE6-#x0AEF] \| [#x0B66-#x0B6F] \| [#x0BE7-#x0BEF] \| [#x0C66-#x0C6F] \| [#x0CE6-#x0CEF] \| [#x0D66-#x0D6F] \| [#x0E50-#x0E59] \| [#x0ED0-#x0ED9] \| [#x0F20-#x0F29] |
| **[89] Extender ::=** #x00B7 \| #x02D0 \| #x02D1 \| #x0387 \| #x0640 \| #x0E46 \| #x0EC6 \| #x3005 \| [#x3031-#x3035] \| [#x309D-#x309E] \| [#x30FC-#x30FE] |

# CHAPTER FIVE
# XML DOCUMENTS

## 5.1 Introduction

XML documents have physical and logical parts. Document's physical structure is the combination of units which are named entities. All XML documents start with the root entity and include logical units below;

- Elements, attributes, processing instructions, comments
- Entity References, CDATA Sections

## 5.2 Character And Markup Data

XML documents hold both data and metadata which is data about data in a structure manner (Harold E.R, 1999). Formatted text is the constitution of character. Units of characters are below;

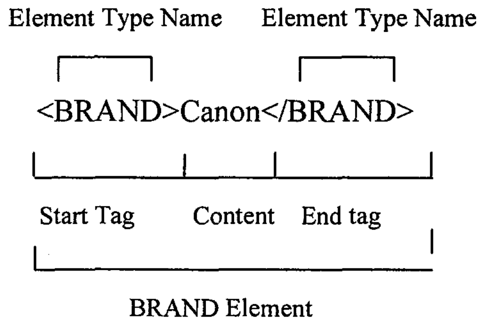
- Alphabetic Unicode letters (English, Arabic, Hebrew and other alphabets)
- Numbers in the range of 0-9 (Europe, Arabic, Persian, Urdu and Egypt numbers and the others)
- Punctuations (; (semicolon), (dot), - (hyphen) and other signs)
- Combination of more than one characters

Characters in XML text format constitute character data and markup data. Markup data specifies document logical structure. Logical structure of an XML document contains start tags, end tags, empty element tags, entity references, character references, comments, CDATA sections, document type declarations (DTD) and processing instructions (Goldfarb C.F., 2001);

- *Start and end tags;* <BRAND>Canon</BRAND> shows start and end tags.
- *Empty elements;* <BRAND NAME='Canon'/>
- *Entity references;* <!ENTITY DEU 'Dokuz Eylul University'> and "&gt;"
- *Character references;* &#x50
- *Comments;* <!-- CAMERA BRAND ELEMENT-->
- *CDATA sections;* Data in this section is cancelled by XML parser. <![CDATA[<BRAND NAME='Canon'/>]]>.
- *Document type declarations;* Elements, attributes and entities are defined in DTDs. <!DOCTYPE category SYSTEM 'category.dtd'>.
- *Processing instructions;* Used for to transfer data to applications using XML processors. <?xml version='1.0' standalone='yes'?>

### 5.3 Elements And Tags

Tags are used with character data and constitute XML elements. In XML documents, there exist three types of tags; *start tag*, *end tag* and *empty element tag*.



BRAND element starts with <BRAND> start tag then Canon content is followed and at the end </BRAND> end tag is occurred. BRAND name is called as *element type name*. Element type names are located between < and > characters. End tags have the / character before element type name. All element type names in XML technology must match some rules;

- It can not start with any combination of *XML* word's upper and lower letters. XML, xML, xmL, xml, XmL, Xml, XMl and xMl words are reserved for future use.
- It can not start with digits (0-9), - (hyphen) character or . (Dot) sign. And can not contain, (comma) character or ! (exclamation) sign
- It can not contain white spaces.
- Element type names are case sensitive. Start and end tag names must be exactly same.

All elements may not contain content in XML documents. Elements which have no content called empty elements (Harold E.R, 1999). The difference of empty elements from start and end tags is the / character. Empty element tag starts with < character, then followed element type name and at the end "/>" string occurs. <BR/> and <HR/> tags are empty tags which are used in HTML language. <BRAND/> or <BRAND> </BRAND> writing techniques are both same.

**Table 5.1 Using empty elements**

```
<CATEGORY NAME="35MM">
  <CAMERA brand="Canon" model="Z155" price="1875000" weight="120 gr." />
</CATEGORY>
```

```
<CATEGORY NAME="35MM">
   <CAMERA brand="Canon" model="Z155" price="1875000" weight="120 gr.">
   </CAMERA>
</CATEGORY>
```

All XML documents have one root element. If we think XML documents as thee structures, root element is at the top of tree and ancestor of all other elements (Goldfarb C.F., 2001). If an <X> element includes <Y> element as child, <X> element is the parent of <Y> and <Y> element <X> is the child of <X>.

**Table 5.2 Valid and non-valid XML type names**

| Valid XML type names | Non-valid XML type names |
|---|---|
| <BRAND><br>*Can contain only letters* | <BRAND 1><br>*Can not contain white-space* |
| <BRAND1><br>*Can contain digits* | <.MARKA><br>*Can not start with dot character* |
| <_BRAND.1><br>*Can start with underline character* | <1.BRAND><br>*Can not start with digits* |
| <_1.BRAND><br>*Can start with underline character* | <BRAND!><br>*Can not contain exclamation* |
| <:BRAND><br>*Can start with colon character* | <-BRAND><br>*Can not start with hyphen* |
| <______><br>*Can contain only underline characters* | <MARKA,1><br>*Can not contain comma character* |

## 5.4 Attributes

Attributes are name-value pairs which are used in start-end and empty element tags (Goldfarb C.F., 2001). Between name and value = character is located. Value of an attribute is written in quotation mark. To read attributes easily in simple text editors, white spaces can be used before and after the occurrence of equal character.

- Attribute names, can not start with any combination of *XML* word's upper and lower letters. XML, xML, xmL, xml, XmL, Xml, XMl and xMl words are reserved for future use.
- Attribute names can not start with digits (0-9), - (hyphen) character or . (Dot) sign. And can not contain , (comma) character or ! (exclamation) sign.
- Attribute names are unique
- Attribute values can not contain < and & characters

**Table 5.3 Valid and non-valid attributes**

| Valid Attributes | Non-valid Attributes |
|---|---|
| name="digital" | name="digital' *Non-suitable quotes* |
| name='digital' | name='digital" *Non-suitable quotes* |
| Oz1="x &amp; y" | oz1="x & y" *& character exists* |
| Oz1="x &lt; y" | Oz1="x < y" *< character exists* |
| Oz1="1234" | oz1=1234 *No quotes exist* |

## 5.5 Processing Instructions

Processing instructions start with <? string and ends with ?> string. Closing string can not be included more than once. The first word after starting string is the name of PI. PI names can not start with digits (0-9), - (hyphen) character or "." (Dot) sign and can not contain "," (comma) character or "!" (exclamation) sign. XML parser does not process PI lines as XML data. Widely used PI types;

- *CSS (Cascade Style Sheets) and XSL references* (WEB_32, 2004)
  CSS and XSL technologies provide representation of XML pages via web browsers. References to style sheets are added XML pages using PI.
  <?xml-stylesheet type="text/css" href="Cameras.css"?>
- *XML declarations*
  XML declarations are one of the most widely used PI type. There is no obligation to write declarations. But if this PI is used, it must be the first line of XML document. Other PIs, white spaces and comment lines can not be added before XML declarations. XML declarations have the *xml* as PI name. *Version*, *standalone* and *encoding* are the attributes of declaration PIs..
  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  - *Version* attribute specifies W3C consortium rules which XML technology must follow. There is two proposed version of XML technology. These versions are "1.0" and "1.1".

- *Standalone* attribute, XML document can get by with no declarations at all. It can also have declarations as part of an internal subset, and it can have declarations in an external subset such as a separate DTD file. "If the value of this attribute is *yes* then there is no possibility to giving references to external documents such as parameter entity references" (WEB_12, 2004). If there is no external reference in an XML document, there is no need to use *standalone* attribute. Although external declarations are not located in XML PI, default value *no* is accepted. The circumstances that value of standalone attribute can not be *no*;
  - If default values of attributes is located outside of the document. (WEB_12, 2004)
  - If XML document have elements or attributes which have external values (WEB_12, 2004).
  - If XML document has entity references except amp, lt, gt, apos and quot (WEB_12, 2004).
- *Encoding* attribute, specifies the encoding characteristic of XML documents. XML parsers accept UTF-8 or UTF-16 encodings as default. Then this encoding is converted Unicode.

**Table 5.4 Supported encodings**

| |
|---|
| ISO-10646-UCS-2, ISO-10646-UCS-4 |
| ISO-2022-JP |
| ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9 |

- *Special PIs to pass parameters to Applications*

  This type of PI is used when special parameters is wanted to pass XML application (desktop, web, wap).

**Table 5.5 Usage of PIs**

```
<?xml version="1.0" encoding="ISO-8859-9" standalone="yes"?>
<!-- CSS reference-->
<?xml-stylesheet type="text/css" href="Cameras.css"?>
<CAMERA_CATEGORIES>
 <CATEGORY NAME="35MM">
  <CAMERA>
   <BRAND>Canon</BRAND>
   <MODEL>Z155</MODEL>
   <PRICE>1875000</PRICE>
   <WEIGHT>120 gr.</WEIGHT>
  </CAMERA>
  <CAMERA>
   <BRAND>Nikon</BRAND>
   <MODEL>S5Z</MODEL>
   <PRICE>1725000</PRICE>
   <WEIGHT>130 gr.</WEIGHT>
  </CAMERA>
 </CATEGORY>
 <CATEGORY NAME="DIGITAL">
  <CAMERA>
   <BRAND>Canon</BRAND>
   <MODEL>EOS 10D</MODEL>
   <PRICE>2249985</PRICE>
   <WEIGHT>135 gr.</WEIGHT>
  </CAMERA>
</CAMERA_CATEGORIES>
<!--Special PIs-->
<?params param1="X" param2="Y"?>
```

**Table 5.6 Valid and non-valid PIs**

| Valid PIs | Non-valid PIs |
|---|---|
| <?delphi version="6.0" param1="X"?> | <? delphi version="6.0" param1="X"?> *There is white space before PI name* |
| <?special PI for Delphi application?> | <?delphi sample!> *Not closed properly* |
| <?xml-stylesheet type="text/css" href="Cameras.css"?> *Acceptable style sheet reference* | <?xml PI line?> *PI name can not be "xml" unless it is a declaration* |

## 5.6 Comments

*"Comments* may appear anywhere in a document outside other markup; in addition, they may appear within the document type declaration at places allowed by the grammar" (WEB_12, 2004). For compatibility, the string "--" (double-hyphen) must not occur within comments.

**Table 5.7 Usage of comments**

```
<?xml version="1.0" encoding="ISO-8859-9" standalone="yes"?>
<!--All  camera categories-->
<CAMERA_CATEGORIES>
   <!-- Cameras of 35MM category -->
   <CATEGORY NAME="35MM">
   </CATEGORY>
   <!-- Cameras of DIJITAL category-->
   <CATEGORY NAME="DIJITAL">
   </CATEGORY>
</CAMERA_CATEGORIES>
<!--End of category list -->
```

**Table 5.8 Valid and non-valid comments**

| Valid Comments | Non-valid Comments |
|---|---|
| <!-- Camera --> | <!--Deniz -- Sibel --><br>*includes -- character* |
| <!--      & Camera &      --> | <!--Deniz and Sibel -->><br>*end character is not valid* |
| <!—Deniz <and/> Sibel --> | |

## 5.7 CDATA Sections

CDATA section is some text to be identified that should escape parsing (WEB_35, 2004). After an XML parser sees the <![CDATA[ sequence that indicates the beginning of a CDATA section and before it sees the ]]> markup that indicates the end, it assumes that all the characters it sees are character data even any left angle brackets and ampersand characters.

*CDATA sections* are used to escape blocks of text containing characters which would other-wise be recognized as markup (WEB_35, 2004). CDATA sections begin with the string *<![CDATA[* and end with the string *]]>*. CDATA sections are popular for showing demonstration XML or HTML markup within an XML document. The markup can be shown as-is with no modifications, but the parser won't confuse the demonstration markup with actual document markup.

**Table 5.9 CDATA usage**

```
<HTMLLESSON>
  <![CDATA[
              <HTML>
              <BODY>
                 <P><B>First lesson about HTML technology...</B></P>
              </BODY>
              </HTML>
   ]]>
</HTMLLESSON>
```

**Table 5.10 Valid and non-valid CDATA sections**

| Valid CDATA Sections | Non-valid CDATA Sections |
|---|---|
| <![CDATA[ <sample> &/......]]> | <![CDATA[ data ]]>.....]]><br>*includes "]]>" string* |
| <XMLLESSON><br><![CDATA[<br>CDATA section starts with <![CDATA[<br>]]><br></XMLLESSON> | |

### 5.8 Entity References

Entities are special XML units which provide to define some internal and external unparsed content like image, video and audio formatted files. References are replaced with their real values during XML documents' parsing processes. 5 predefined entity references are defined in XML technology.

**Table 5.11 Predefined entities (WEB_12, 2004)**

| Entity | Description |
|---|---|
| &amp; | & (and) character. Used as the first character of entity references. |
| &lt; | < character. It is the starting character of XML element's tags. |
| &gt; | > character. It is the closing character of XML element's tags. |
| &quot; | "(quote) character. Used for to keep XML attribute values. |
| &apos; | ' (apos) character. Used for to keep XML attribute values. |

If instead of reference usage predefined entity references' real values are used within XML documents outside of CDATA sections, they can be realized as XML markup. So they can cause error.

**Table 5.12 Entity reference declaration and usage**

```
<?xml version="1.0" encoding="iso-8859-9"?>
 <!DOCTYPE SCHOOL [
 <!ENTITY deu  "Dokuz Eylül University">
 <!ELEMENT SCHOOL (NAME)>
 <!ELEMENT NAME (#PCDATA)>
<!-- DTD Ends-->
 ]>

<SCHOOL>
    <NAME>&deu;</NAME>
</SCHOOL>
```

Entity references are defined in the DTD files. There exists a prototyping mechanism which defines the logical structure of XML document. DTD files also define the elements, attributes and notations in addition to entities.

## 5.9 Well-Formed XML Documents

To parse and process XML documents, documents must meet the well-formed rules and constraints at least (Harold E.R, 1999). Some of the well-formalness rules defined for XML 1.0 technology is below.

- XML document has only one root element that includes all other sub and child elements (WEB_12, 2004).

**Table 5.13 Root element**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<!-- DTD declaration-->
<ROOT_ELEMENT>
  <CHILD>Content</CHILD>
</ROOT_ELEMENT>
<?Processing Instruction1?>
```

- All start tags must be ended with a suitable closing tag. Element type names within start and end tag must be same sensitively (WEB_12, 2004).

**Table 5.13 Starting and closing element**

```
<CHILD> Content Info </CHILD>
```

- Elements must be nested properly in the tree hierarchy. One element can contain other elements. Sub-elements start and end tags must be opened and closed with the proper order (WEB_12, 2004).

**Table 5.14 Nesting**

| **Well Nested** |
|---|
| <TOP> <BOTTOM> content </BOTTOM></TOP> |
| **Not Well Nested** |
| <TOP>  <BOTTOM> content </TOP> </BOTTOM> |

- Parameter entity declarations in the internal DTDs must include the whole definition of element declaration (WEB_12, 2004).

**Table 5.15 Parameter entity references**

```
<?xml version="1.0" encoding="iso-8859-9"?>
 <!DOCTYPE SCHOOL [
<!ELEMENT OKUL (NAME,CITY)>
<!--Since it hold the whole declaration of element, it is a valid
parameter entity definition -->
<!ENTITY % nameDecl "<!ELEMENT NAME (#PCDATA)>">
  %nameDecl;
<!--Since it hold the part of declaration of element, it is not a valid
parameter entity definition -->
   <!ENTITY % cityDecl "CITY (#PCDATA)">
   <!ELEMENT %cityDecl ;>
-->
<!ELEMENT CITY (#PCDATA)>
]>
<SCHOOL>
    <NAME>Dokuz Eylül University</NAME>
    <CITY>İzmir</CITY>
</SCHOOL>
```

- Attribute names are unique and can not use more than once (WEB_12, 2004).
- Attribute values can not include external entity references (WEB_12, 2004).
- In a standalone document declaration, the value "yes" indicates that there are no markup declarations external to the document entity (either in the DTD external subset, or in an external parameter entity referenced from the internal subset) which affect the information passed from the XML processor to the application (WEB_12, 2004).

# CHAPTER SIX
# DTD TECHNOLOGY

### 6.1 Introduction

Document type definition is a mechanism that specifies XML document's logical structure and controls document's markup units for validation purposes (WEB_34, 2004). Many predefined DTD exists for many sectors. If you want you can use these predefined DTDs or you can create your own DTD.

On the contrary of XSL, XPath and XML Schema technologies, DTD mechanism is created using EBNF like regular expressions (WEB_33, 2004). The sub-topics that can be researched under DTD are below;

- Reading, interpreting and creating DTDs.
- Valid XML Document, Internal and external DTDs
- Element Declarations, Attribute Declarations
- Entity Declarations, Conditional Sections
- Language Declarations, Validation Constraints

### 6.2 Valid Documents

An XML document must wholly match the well-formedness rules and constraints of W3C to be able to parse and process by browsers. Document's validity depends on well-formalness. An XML document must be firstly well-formed for validation (WEB_12, 2004). A non-well formed XML document can not be valid.

"An XML document is called "Valid Document" if and only if it is well-formed and has a DTD or schema mechanism" (WEB_12, 2004). It is possible to control XML element and attribute types and their contents.

## 6.3 Document Type Declaration

DTD mechanism specifies the XML document structure by providing declaration of markup units. Working logic of DTD seems header files in C programming language which defines the prototypes of functions and variables. Three types of DTD can use with XML technology (WEB_36, 2004);

- *Internal DTD* is defined in XML documents. It is used to define only related XML document's markup units. The other XML documents can not use these declarations. It is easy to distribute and test internal DTDs. But they can not use commonly by more than one XML documents (Pitts N., 2004).
- *External DTD* is defined external file with the extension of *.dtd* and referenced from XML document. This type of DTD is can be used more than one XML documents (WEB_36, 2004). When DTD changes, it is not necessary to change XML document. Two types of external DTD exists;
  - General External DTD
  - Special External DTD
- *Using internal and external DTD together*, (Pitts N., 2004) with this method both DTD can use commonly within more than one document and personalized according to each XML document.

When number of elements and attributes are increased, document control is got hard. At this point, a DTD mechanism is needed which will define document logical structure. XML units that can use in DTD mechanism are below;

**Table 6.1 DTD units (Pitts N., 2004)**

| DTD Unit | Objective |
|---|---|
| ELEMENT | Defines elements' orders, content and content types. |
| ATTLIST | Defines attributes' names, types, default values and containment sub or child elements. |
| ENTITY | Provides to use different types of parsed or unparsed content. |
| NOTATION | Defines unparsed audio, image and video formats. |
| Processing Instructions | It is used for XML declarations and special purposes. |
| Comment Lines | It is used to give some information and attention. |

### 6.3.1 Internal DTD

DTD declaration starts with *<!DOCTYPE* string literal, then followed by root element name.

<!DOCTYPE *root_element* [
*dtd_declarations*
]>

Root element name must be presented within mechanism. After root element name, *[* character is followed, then DTD declarations (element, attribute, entity) are defined. And finally, string literal *]>* is located. DOCTYPE declaration binds DTD mechanism and XML document. It is located between xml declaration and root element. Processing instructions or comment lines can be located before DOCTYPE declaration. General structure of DTD mechanism is below;

**Table 6.2 Simple DTD architecture**

```
                   <?xml version="1.0"?>        ———→ Xml declaration
                   <!DOCTYPE SAMPLE         DTD root element name
Starting Char.  [      <!ELEMENT SAMPLE (#PCDATA) > Element Type N.
Closing Char   ]>
String                 <SAMPLE>A SIMPLE SAMPLE</SAMPLE>
                          ↓
                       Root element
```

In table 6.2, It is showed that a simple DTD architecture in an XML document. According to declaration XML document constituted one element. SAMPLE is the name of root element and can only include character data.

**Table 6.3 Simple DTD usage**

```
<!DOCTYPE CAMERA_CATEGORIES [
<!ELEMENT CAMERA_CATEGORIES (CATEGORY+)>
<!ELEMENT CATEGORY (NAME, CAMERA+)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT CAMERA (BRAND,MODEL,PRICE,WEIGHT)>
<!ELEMENT BRAND (#PCDATA)>
<!ELEMENT MODEL (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT WEIGHT (#PCDATA)>
]>
```

In table 6.3, a dtd declaration is created which includes camera categories and their cameras. There exist 8 elements which are CAMERA_CATEGORIES, CATEGORY, NAME, CAMERA, BRAND, MODEL, PRICE and WEIGHT. The root element is CAMERA_CATEGORIES which includes CATEGORY elements. CATEGORY element has NAME and CAMERA element. And CAMERA element has BRAND, MODEL, PRICE and WEIGHT child elements. Child elements are #PCDATA type which can contain parsed character data.

**Table 6.4 DTD comparing**

| **[DTD1]** |
|---|
| <!DOCTYPE CAMERA_CATEGORIES [ |
| <!ELEMENT WEIGHT (#PCDATA)> |
| <!ELEMENT PRICE (#PCDATA)> |
| <!ELEMENT MODEL (#PCDATA)> |
| <!ELEMENT BRAND (#PCDATA)> |
| <!ELEMENT CAMERA (BRAND,MODEL,PRICE,WEIGHT)> |
| <!ELEMENT NAME (#PCDATA)> |
| <!ELEMENT CATEGORI (NAME, CAMERA+)> |
| <!ELEMENT CAMERA_CATEGORIES (CATEGORY+)> |
| ]> |
| **[DTD2]** |
| <!DOCTYPE CAMERA_CATEGORIES [ |
| <!ELEMENT CAMERA (BRAND,MODEL,PRICE,WEIGHT)> |
| <!ELEMENT PRICE (#PCDATA)> |
| <!ELEMENT MODEL (#PCDATA)> |
| <!ELEMENT CATEGORY (NAME, CAMERA+)> |
| <!ELEMENT WEIGT (#PCDATA)> |
| <!ELEMENT CAMERA_CATEGORIES (CATEGORY+)> |
| <!ELEMENT BRAND (#PCDATA)> |
| <!ELEMENT NAME (#PCDATA)> |
| ]> |

In table 6.4, row orders are changed in DTD. Technically, it is not different from the DTD in table 6.3. The important point that the element declarations, element names and content order is not changed.

**Table 6.5 Element Orders**

| **[Order1]** |
|---|
| <!ELEMENT CAMERA (BRAND,MODEL,PRICE,WEIGHT)> |
| **[Order2]** |
| <!ELEMENT CAMERA (WEIGHT, BRAND, PRICE, MODEL)> |

In table 6.5, child elements' orders are changed and technically these elements must match this order. Otherwise, XML document that has this DTD will not be a valid document.

**Table 6.6 Internal DTD Usage**

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE CAMERA_CATEGORIES [
<!ELEMENT CAMERA_CATEGORIES (CATEGORY+)>
<!ELEMENT CATEGORY (NAME, CAMERA+)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT CAMERA (BRAND,MODEL,PRICE,WEIGHT)>
<!ELEMENT BRAND (#PCDATA)>
<!ELEMENT MODEL (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT WEIGHT (#PCDATA)>
]>
<CAMERA_CATEGORIES>
 <CATEGORY>
  <NAME>35MM</NAME>
  <CAMERA>
   <BRAND>Canon</BRAND>
   <MODEL>Z155</MODEL>
   <PRICE>1875000</PRICE>
   <WEIGHT>120 gr.</WEIGHT>
  </CAMERA>
  <CAMERA>
   <BRAND>Nikon</BRAND>
   <MODEL>S5Z</MODEL>
   <PRICE>1725000</PRICE>
   <WEIGHT>130 gr.</WEIGHT>
  </CAMERA>
 </CATEGORY>
</CAMERA_CATEGORIES>
```

In table 6.6, it is showed an internal DTD declaration which is in camera categories XML. DTD declaration follows after XML declaration and DOCTYPE name is same as *KAMERA_KATEGORILERI.*

### 6.3.2 External DTD

External DTDs must match some rules and declarations like internal DTDs. If a reference from XML documents to external DTDs is constructed an URI (Uniform Resource Identifier) is used (WEB_36, 2004). URI is a notation for naming resources on the Web. An URL (Uniform Resource Locator) such as *http://www.deu.edu.tr* is one kind of URI. An XML processor treats a relative URI as being relative to the entity where it's stored, not relative to the document entity ultimately containing the reference.

Standalone attribute of XML declaration which has external DTD reference has another importance. Because the value *yes* indicates that there are no markup declarations external to the document entity and the value *no* indicates that there are or may be such external markup declarations. So the value of this attribute must be *no* within external DTDs.

**Table 6.7 XML declaration for external DTD**

```
<?xml version="1.0" standalone="no"?>
```

Two types of external DTDs exist (Pitts N., 2004). *General External DTD*, commonly used standard DTDs which are used by sectors and associations. Financial, transport or academic associations generally uses this type of DTDs. *Special External DTD*, are created for special purposes by persons or societies (WEB_36, 2004).

### 6.3.3 General External DTD

General external DTDs are constructed using PUBLIC literal string within DOCTYPE declaration (Pitts N., 2004). General DTDs have official FPIs (Formal Public Identifier) (WEB_37, 2004). FPIs can be thought as keys over some servers to control DTDs versions.

**Table 6.8 XML 1.0 version DTD declaration (WEB_38, 2004)**

```
<!DOCTYPE spec PUBLIC "-//W3C//DTD Specification V2.0//WIDTH"
                          "/XML/1998/06/xmlspec-v20.dtd">
```

In table 6.8, an external DTD reference declared for DTD 2.0 version is showed. Reference starts with *<!DOCTYPE* string literal, then followed root element *spec*.

*PUBLIC* keyword which comes after *PUBLIC* string is the keyword used in references to general external DTDs. Reference includes two types of information;

1. *DTD owner info*, "-//W3C//DTD Specification V2.0//WIDTH" (WEB_38, 2004)

- Starts with - character. This character tells that the reference is not a common standard accepted by ISO like associations (WEB_38, 2004).
- If plus character + was located instead of - it would tell that the reference is a common standard accepted associations (WEB_38, 2004).
- **W3C** (World Wide Web Consortium), is the owner association of DTD, all rights reserved for W3C and DTD has been developed by W3C (WEB_38, 2004).
- **DTD Specification V2.0**, is the defining name of DTD file. This can contain all characters except // literal (WEB_38, 2004).
- **WIDTH**, defines the language of xml which includes DTD. These two character codes are named ISO639 language codes. WIDTH shows that the language is English (WEB_38, 2004).

**Table 6.9 ISO639 codes**

| |
|---|
| En (English) |
| Tr (Turkish) |
| TR (Turkish) |
| FR (French) |

In table 6.9, some ISO639 standard language codes are located. It is possible to research all codes from *http://lcweb.loc.gov/standards/iso639-2/bibcodes.html URL* address. Now, 2704 defined language code exists.

2. *Location information of DTD*, "/XML/1998/06/xmlspec-v20.dtd" (WEB_38, 2004)

Location information is defined in URL (Uniform Resource Locator) standard. Address tells us DTD definition of XML 1.0 technology is saved in a *.dtd* extension file which has the name *xmlspec-v20.dtd*. Extension may not have been

*.dtd*, but for common usage and easiness purposes this naming convention is important.

**Table 6.10 External general DTD usage**

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE CAMERA_CATEGORIES
                 PUBLIC "-//CameraWorld//Camera Definition 1.0//TR"
                        "http://www.CameraWorld.com.tr/CamSpec.dtd">
<CAMERA_CATEGORIES>
 <CATEGORY> <NAME>35MM</NAME>
  <CAMERA>
   <BRAND>Canon</BRAND>
   <MODEL>Z155</MODEL>
   <PRICE>1875000</PRICE>
   <WEIGHT>120 gr.</WEIGHT>
  </CAMERA>
  <CAMERA>
   <BRAND>Nikon</BRAND>
   <MODEL>S5Z</MODEL>
   <PRICE>1725000</PRICE>
   <WEIGT>130 gr.</WEIGHT>
  </CAMERA>
 </CATEGORY>
 <CATEGORY> <NAME>DIJITAL</NAME>
  <CAMERA>
   <BRAND>Canon</BRAND>
   <MODEL>EOS 10D</MODEL>
   <PRICE>2249985</PROCE>
   <WEIGHT>135 gr.</WEIGHT>
  </CAMERA>
 </CATEGORY>
</CAMERA_CATEGORIES>
```

In table 6.10, an XML document which has general external DTD is shoed. Document's root element name *CAMERA_CATEGORIES* is same as dtd declaration name and *PUBLIC* keyword is used to specify the reference as general. Owner information of DTD is a URN address and it tells us that the DTD is not a common standard accepted associations, because it is started with the - character.

*CameraWorld* is the name of owner association and *Camera Definition 1.0* is the defining name of DTD. *TR* code specifies the language *Turkish*. *http://www.CameraWorld.com.tr/CamSpec.dtd* is the URL address maps the physical location of DTD.

### 6.3.4 Special External DTD

These types of DTDs are created for special purposes by persons or societies. They are not open to common usage and not general standards.

**Table 6.11 Special external DTD**

```
<!DOCTYPE CAMERA_CATEGORIES SYSTEM
              "http://www.KibeleNet.com/CamKat.dtd">
```

In table 6.11, a reference is showed which is defined a DTD file for camera categories file. Reference starts with *<!DOCTYPE* literal and is followed with root element name CAMERA_CATEGORIES that is the name of root element. *http://www.KibeleNet.com/ KamKat.dtd* address is the URL (Uniform Resource Locator) location information. *SYSTEM*, is the keyword for special external DTD files.

**Table 6.12 External special DTD usage**

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE CAMERA_CATEGORIES SYSTEM
              "CamKat.dtd">
<CAMERA_CATEGORIES>
  <CATEGORY>
    <NAME>35MM</NAME>
    <CAMERA>
      <BRAND>Canon</BRAND>
      <MODEL>Z155</MODEL>
      <PRICE>1875000</PRICE>
      <WEIGHT>120 gr.</WEIGHT>
    </CAMERA>
  </CATEGORY>
  <CATEGORY>
```

**Table 6.12 Continued...**

```
   <NAME>DIJITAL</NAME>
   <CAMERA>
    <BRAND>Canon</BRAND>
    <MODEL>EOS 10D</MODEL>
    <PRICE>2249985</PRICE>
    <WEIGHT>135 gr.</WEIGHT>
   </CAMERA>
 </CATEGORY>
</CAMERA_CATEGORIES>
```

In table 6.12, an XML document is showed which special external DTD. Location of document's DTD location address is specified using local file path instead of URL web address. According to file system path DTD file must be located same folder with XML document.

Both in general DTDs and in special DTDs one of the most important point is validity of the location address (URL web address or file path). If DTD can not be found by XML parser, DTD processing error is occurred.

### 6.3.5 Using External And Internal DTDs Together

Using external and internal DTDs together, both DTD can be used commonly within more than one document and personalized according to each XML document. External DTDs are referenced using "SYSTEM" literal and then internal DTD definition begins. Internal DTD definition starts with *[* character and ends with *]>* literal (Pitts N., 2004).

**Table 6.13 External and internal DTD usage**

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE CAMERA_CATEGORIES SYSTEM "Parent.dtd" [
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT BRAND (#PCDATA)>
<!ELEMENT MODEL (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT WEIGHT (#PCDATA)>
```

**Table 6.13 Continued...**

```
]>
<CAMERA_CATEGORIES>
 <CATEGORY>
  <NAME>35MM</NAME>
  <CAMERA>
   <BRAND>Canon</BRAND>
   <MODEL>Z155</MODEL>
   <PRICE>1875000</PRICE>
   <WEIGHT>120 gr.</WEIGHT>
  </CAMERA>
 </CATEGORY>
</CAMERA_CATEGORIES>
```

In table 6.13, an XML document is showed which has both internal DTD and external DTD. External DTD is referenced using SYSTEM keyword. "Parent.dtd" is the external DTD file includes parent element declarations and internal DTD includes child element declarations.

If both external and internal subsets are used, the internal subset is considered to occur before the external subset (Pitts N., 2004). This has the effect that entity and attribute-list declarations in the internal subset take precedence over those in the external subset.

**Table 6.14 Entities in external and internal DTDs**

| **[INT.DTD]** |
|---|
| <?xml version="1.0" encoding="iso-8859-9"?><br><!DOCTYPE SCHOOL SYSTEM "ext.DTD" [<br><!ELEMENT SCHOOL (#PCDATA)><br><!ENTITY schoolname "DEU"><br>]><br><SCHOOL>&schoolname;</SCHOOL> |
| **EXT.DTD** |
| <!ENTITY schoolname "Dokuz Eylul University"> |

In table 6.14, *schoolname* entity reference is used both in internal DTD and in external DTD. Since the priority of internal DTD, SCHOOL element will has *DEU* content.

## 6.4 Element Declarations

All element type declarations in an XML documents must be defined within DTD files for validation purposes. It is possible to define element name and content type using element type declarations. The general structure of element type declaration is below.

<!ELEMENT *elementName* *contentDefinition*>

Element declaration starts with "<!ELEMENT" literal, then followed white-space character, followed element name, followed content descriptor, optional white-space character and ends with ">" character (Pitts N., 2004). Element type name must match some constraints;

- It can not start with any combination of *"XML"* word's upper and lower letters. XML, xML, xmL, xml, XmL, Xml, XMl and xMl words are reserved for future use.
- It can not start with digits (0-9), "-" (hyphen) character or "." (Dot) sign. And can not contain "," (comma) character or "!" (exclamation) sign
- It can not contain white spaces.

Element type name has 5 types of content; normal content, only character data, mixed content, any kind of content type, empty elements.

### 6.4.1 Elements with Normal Content

They include one or more sub element. Technically, an element type which has only one sub element is called base element (Pitts N., 2004).

**Table 6.15 Elements with normal content**

```
<!ELEMENT PERSONNEL (PERSONNEL_INFO)>
```

In table 6.15, PERSONNEL element which has normal content is declared. This element has only one element called PERSONNEL_INFO. PERSONNEL element does not include other sub elements (except PERSONNEL_INFO element) or character data. It does not mean that if PERSONNEL element declaration exists, automatically PERSONNEL_INFO element is declared.

**Table 6.16 Valid and non-valid elements with normal content**

| **Valid document** |
|---|
| <PERSONNEL><br>  <PERSONNEL_INFO>01-Ahmet Tan</PERSONNEL_INFO><br></PERSONNEL> |
| **Non-valid document** |
| <PERSONNEL><br>  <PERSONNEL_INFO>01-Ahmet Tan</PERSONNEL_INFO><br>  <PERSONNEL_INFO>02-Sibel KOPARAN</PERSONNEL_INFO><br></PERSONNEL> |

In table 6.16, PERSONNEL element's two types of usage in XML documents are showed. First usage is a valid. It has only one sub element called PERSONNEL_INFO and does not contain any character data or sub element. Second usage is not valid, since PERSONNEL element two PERSONNEL_INFO elements.

Elements with normal content can also contain more than one element. Sub elements are located in parenthesis with commas and with ordering.

**Table 6.17 Elements have sub-elements with normal content**

```
<!ELEMENT PERSONNEL (PERSONENLNO, PERSONNELNAME)>
```

In table 6.17, defined PERSONNEL element has normal content and has two sub elements. These sub elements are also child elements which are named PERSONENLNO and PERSONNELNAME.

### 6.4.2 Elements Has Only Character Data

Elements which have only character data are called child element. These elements contain parsed character data (Pitts N., 2004). Parsed character data can not include "<", "&", ">" characters, but can include character references which are mapped to ignored characters "&lt;", "&amp;" and "&gt;"

**Table 6.18 Elements have only character data**

```
<!ELEMENT BRAND (#PCDATA)>
```

In table 6.18, BRAND element is declared which has only character data. BRAND element can not contain any elements.

### 6.4.3 Elements Has Mixed Content

Elements which have mixed content may have both parsed character data and other sub elements (Pitts N., 2004).

**Table 6.19 Elements have mixed content**

```
<!ELEMENT PERSONNEL (#PCDATA | PERSONNEL_INFO)*>
```

In table 6.19, an element is showed which has character data and child element PERSONNEL_INFO. Pipe character in the content declaration means logical OR and PERSONNEL element can contain character data OR PERSONNEL_INFO child element. "*" character outside of the right parenthesis means content in the parenthesis can be used in XML document zero or more times. But it is not possible to say the exact number of occurrences of child elements.

**Table 6.20 Valid and non-valid elements with mixed content**

| **Valid document** |
|---|
| <PERSONNEL> |
| 0001-SOFTWARE |
| <PERSONNEL_INFO>Ahmet Tan-İZMİR</PERSONNEL_INFO> |
| </PERSONNEL> |
| **Valid document** |
| <PERSONNEL> |
| 0001-SOFTWARE |
| <PERSONNEL_INFO>Göksel ÜÇER-İZMİR</PERSONNEL_INFO> |
| <PERSONNEL_INFO>Ahmet Tan-İZMİR</PERSONNEL_INFO> |
| </PERSONNEL> |
| **Non-valid document** |
| <PERSONNEL> |
| <PERSONNEL_INFO>Ahmet Tan-İZMİR</PERSONNEL_INFO> |
| </PERSONNEL> |

An element which has mixed content at least must have the declaration in table 6.20. Below element type declarations do not match the mixed content element type.

- Element declaration which has mixed content must start with #PCDATA keyword before other sub elements.

  *<!ELEMENT PERSONNEL (PERSONNEL_INFO | #PCDATA)*>,*

- Element declaration which has mixed content can not include "," (comma) character instead of "|" (pipe) character between sub element #PCDATA string literal.

  *<!ELEMENT PERSONNEL (PERSONNEL_INFO , #PCDATA)*>,*

- Element declaration which has mixed content can not specify sub elements number of occurrences.

  *<!ELEMENT PERSONNEL (PERSONNEL_INFO , #PCDATA)>,*

### 6.4.4 Elements Have Any Kind Of Content Type

If an element is wanted to define with capability that can contain any kind of content, must be declared using *ANY* keyword within DTD file (Pitts N., 2004).

**Table 6.21 Elements have any kind of content type**

| |
|---|
| `<!ELEMENT PERSONNEL (ANY)>` |

PERSONNEL element in table 6.21 may have normal content, only character data or mixed content. These elements can be thought as joker elements. When you design XML applications, in first step defining elements in ANY can be easier and decrease your development time. But you can forget that element. In real life applications although it is technically possible defining ANY types, it would not be realistic an element with ANY type.

**Table 6.22 Valid and non-valid elements with any kind of content**

| **Valid document** |
|---|
| `<PERSONNEL>`<br>  `<ID>0001</ID>`<br>  `<NAME>Sibel KOPARAN</NAME>`<br>`</PERSONNEL>` |
| **Non-valid document** |
| `<PERSONNEL>`<br>  `0001-Sibel KOPARAN`<br>`</PERSONNEL>` |

In table 6.22, some content of PERSONNEL element which is defined with mixed content.

### 6.4.5 Empty Elements

Empty elements have no content. The difference of empty elements from start and end tags is the / character. Empty element tag starts with < character, then followed element type name and at the end /> string occurs. When an element is wanted define as empty element, *EMPTY* keyword must be used in DTDs (Pitts N., 2004).

**Table 6.23 Empty elements**

| |
|---|
| <!ELEMENT PERSONNEL EMPTY> |

In table 6.23, PERSONNEL element has an element declaration as empty element using EMPTY keyword. This element can be showed in an xml document like below.

**(1)** **(2)**

*<PERSONNEL></PERSONNEL>* *or* *<PERSONNEL/>*

These two declarations are same technically. But in second empty element declaration usage must be important to distinguish normal XML elements from empty XML elements. Empty elements can not have any content but it does not mean they can not have attributes. In real life, empty elements generally used with attributes like <IMG> HTML element.

**Table 6.24 Valid empty elements**

| **Valid document** |
|---|
| <PERSONNEL ID="0001" NAME="Sibel KOPARAN"> </PERSONNEL> |
| **Valid document** |
| <PERSONNEL ID="0001" NAME="Sibel KOPARAN"/> |

In table 6.24, empty PERSONNEL element is showed in two different XML declarations. Second XML declaration is more clear (more readable and distinguished from normal elements easily) than first one.

## 6.5 Element Declarations

"BNF is an abbreviation for Backus-Naur-Form. BNF grammars are an outgrowth of compiler theory. A BNF grammar defines what is and is not a syntactically correct or not" (WEB_10, 2004).

**Table 6.25 XML 1.0 first EBNF rule**

| |
|---|
| **[1] document ::=** (prolog element Misc*) |

In table 6.25, according to rule, an XML document must start *prolog* element. Simply, a prolog element includes DTD and XML declaration definitions. A root element must follow the "Prolog" element. Third non-terminal "*Misc*" element is optional. *Misc* element includes white-spaces, comments or processing instructions.

**Table 6.26 Element order and number identifiers (Harold E.R, 1999)**

| Identifier | Description |
|---|---|
| \| | Means logical OR operation |
| , | Means logical AND operation |
| ( ) | Groups elements |
| A* | Zero or more occurrences of A |
| A+ | One or more occurrences of A |
| A? | Zero or one occurrences of A |
| A B | A comes after B |
| A \| B | Matches A or B but not both |

## 6.6 Order Identifiers

Order identifier character (comma character), specifies the exact order of sub elements or child elements. Using this identifier you set the occurrence order of element. XML elements' orders are very important for errorless XML parsing (Pitts N., 2004). So order identifiers within DTD files are same importance.

**Table 6.27 Element declaration using order identifiers**

```
<!ELEMENT CAMERA (BRAND, MODEL, PRICE, WEIGHT)
                  1.      2.      3.      4.
```

In table 6.27, a CAMERA element with 4 sub elements is defined. Sub elements are located in an exact order within CAMERA element structure. In first order BRAND element, in second order MODEL element, in third order PRICE element and finally in forth order WEIGHT element follows.

When this CAMERA element is used in an XML document, element's order must be same as above. Otherwise the XML document will not be valid.

**Table 6.28 XML element usage**

**Valid ordered element usage**

```
<CAMERA>
  <BRAND>Canon</BRAND>
  <MODEL>Z155</MODEL>
  <PRICE>1875000</PRICE>
  <WEIGHT>120 gr.</WEIGHT>
</CAMERA>
```

**Non-valid ordered element usage**

```
<CAMERA>
  <BRAND>Canon</BRAND>
  <MODEL>Z155</MODEL>
  <WEIGHT>120 gr.</WEIGHT>
  <PRICE>1875000</PRICE>
</CAMERA>
```

In table 6.28, two CAMERA element usage is showed. Although first usage is valid, the second one is not valid. Second CAMERA element's children PRICE and WEIGHT elements' orders are changed.

When using order identifiers, the only way to change the elements' orders is using optional occurrence character *, at most one occurrence character ?, at least one occurrence character (WEB_10, 2004). For example in a real world XML example which includes camera information as content, knowing all camera's weight can be impossible. So WEIGHT element is wanted to be skipped in some cases. In a such application CAMERA element can be declared in two ways;

1. <!ELEMENT CAMERA (BRAND, MODEL, PRICE, (WEIGHT)*)
2. <!ELEMENT CAMERA (BRAND, MODEL, PRICE, (WEIGHT)?)

In first declaration WEIGHT element can be used optional, it means zero or more occurrences of element is possible and legal. But in second declaration WEIGHT element can be use at most once.

**Table 6.29 XML element usage with order and number identifiers**

```
<CAMERA>
 <BRAND>Canon</BRAND>
 <MODEL>Z155</MODEL>
 <PRICE>1875000</PRICE>
</CAMERA>
```

In table 6.29, CAMERA element and its sub elements matches both first declaration and second declaration.

The other way to change the order of elements is repeated of sub elements. For example in a real world XML example which includes camera information as content, it can be wanted to define two types of prices for franchisers and customers. In a such application CAMERA element can be declared in two ways;

1. <!ELEMENT CAMERA (BRAND, MODEL, (PRICE)*, WEIGHT)
2. <!ELEMENT CAMERA (BRAND, MODEL, (PRICE)+, WEIGHT)

**Table 6.30 Order and number identifiers**

```
<CAMERA>
 <BRAND>Canon</BRAND>
```

**Table 6.30 Continued...**

```
    <MODEL>Z155</MODEL>
    <PRICE>18750000</PRICE> <!--Franchiser Price-->
    <PRICE>20000000</PRICE> <!--Customer Price-->
    <WEIGHT>120 gr.</WEIGHT>
  </CAMERA>
```

An element which is not defined in DTD mechanism can not be used as an element in XML documents.

**Table 6.31 Sub element usage that is not defined**

```
<CAMERA>
  <BRAND>Canon</BRAND>
  <MODEL>Z155</MODEL>
  <PRICE>18750000</PRICE>
  <WEIGHT>120 gr.</WEIGHT>
  <FOCUS>4x8</FOCUS>
</CAMERA>
```

## 6.7 Alternative Identifiers

Alternative identifiers let elements to be located selectively instead of defining an exact order. Pipe character "|" is used as an alternative identifier (WEB_10, 2004). By using this character between elements or element groups, it is provided to define alternative element declarations.

**Table 6.32 Element declaration using alternative identifier**

```
<!ELEMENT PERSONNEL (ID | NAME)>
```

In table 6.32, according to PERSONNEL element, it may have one of the ID or NAME sub elements.

**Table 6.33 Element declaration using alternative identifier**

| **Valid element usage** |
|---|
| &lt;PERSONNEL&gt;<br>  &lt;ID&gt;0001&lt;/ID&gt;<br>&lt;/PERSONNEL&gt; |
| **Valid element usage** |
| &lt;PERSONNEL&gt;<br>  &lt;NAME&gt;Sibel KOPARAN&lt;/NAME&gt;<br>&lt;/PERSONNEL&gt; |
| **Non-valid element usage** |
| &lt;PERSONNEL&gt;<br>  &lt;ID&gt;0001&lt;/ID&gt;<br>  &lt;NAME&gt;Sibel KOPARAN&lt;/NAME&gt;<br>&lt;/PERSONNEL&gt; |

## 6.8 Using Order and Alternative Identifiers By Grouping

If it is needed to group some elements which are in same concept, parenthesises are used for grouping. By grouping technique, it is possible to use orders within alternatives or alternatives within orders. It is also possible to assign "?", "+" and "*" characters to groups (WEB_10, 2004).

**Table 6.34 Defining elements using order and alternative identifiers by grouping**

```
<!ELEMENT PERSONNEL (ID, NAME, (PHONE | FAX | EMAIL) )>
```

PERSONNEL element defined in table 6.34, can be formed as an ID element and a NAME element and one of the PHONE, FAX or EMAIL element. The last three elements are grouped alternatively. PERSONNEL element can not contain two or three of these elements at the same time.

**Table 6.35 Using order and alternative identifiers by grouping**

| **Valid element usage** |
|---|
| &lt;PERSONNEL&gt;<br>  &lt;ID&gt;0002&lt;/ID&gt;<br>  &lt;NAME&gt;Ahmet Tan&lt;/NAME&gt;<br>  &lt;PHONE&gt;02323457869&lt;/PHONE&gt; |

| **Table 6.35 Continued...** |
|---|
| </PERSONNEL> |
| **Valid element usage** |
| <PERSONNEL> |
|   <ID>0002</ID> |
|   <NAME> Ahmet Tan </NAME> |
|   <EMAIL>dkilinc@deu.edu.tr</EMAIL> |
| </PERSONNEL> |
| **Non-valid element usage** |
| <PERSONNEL> |
|   <ID>0002</ID> |
|   <NAME> Ahmet Tan </NAME> |
| </PERSONNEL> |
| **Non-valid element usage** |
| <PERSONNEL> |
|   <ID>0002</ID> |
|   <NAME> Ahmet Tan</NAME> |
|   <PHONE>02323457869</PHONE> |
|   <EMAIL>dkilinc@deu.edu.tr</EMAIL> |
| </PERSONNEL> |

In table 6.35, valid and non-valid PERSONNEL XML elements are showed. In the first non-valid PERSONNEL element one of the PHONE, FAX or EMAIL elements is not used. And in the second non-valid PERSONNEL element PHONE and EMAIL elements are used together.

**Table 6.36 Using order and alternative identifiers by grouping**

```
<!ELEMENT PERSONNEL ( (ID) | (NAME, (PHONE)+ , (FAX)* , (EMAIL)+) )>
```

In table 6.36, PERSONNEL element can be created in two ways in an XML document. In the first PERSONNEL element can be formed as using only one ID element. In the second PERSONNEL element can be formed as one NAME element, one or more PHONE element, zero or more FAX element and one or more EMAIL element.

**Table 6.37 Using order and alternative identifiers by grouping (2)**

**Valid element usage**

```
<PERSONNEL>
  <ID>0002</ID>
</PERSONNEL>
```

**Valid element usage**

```
<PERSONNEL>
  <NAME> Ahmet Tan</NAME>
  <PHONE>02327659334</PHONE>
  <PHONE>05324853032</PHONE>
  <EMAIL>dkilinc@deu.edu.tr</EMAIL>
</PERSONNEL>
```

**Non-valid element usage**

```
<PERSONNEL>
  <ID>0002</ID>
  <NAME> Ahmet Tan</NAME>
</PERSONNEL>
```

**Non-valid element usage**

```
<PERSONNEL>
  <NAME> Ahmet Tan</NAME>
  <PHONE>02323457869</PHONE>
  <FAX>02323457769</FAX>
</PERSONNEL>
```

In table 6.37, valid and non-valid PERSONNEL XML elements are showed. In the first non-valid element ID and NAME elements are located together. In the second non-valid element EMAIL element is not used.

## 6.9 Specifying Exact Number of Elements in DTDs

In DTD technology, it is not possible to give constraints directly to number occurrences like at least 3 times, at most 5 times. Some occurrence constraints that can be applied to elements; exactly one element, at least one element, at most one element, optional number of occurrences of element (Pitts N., 2004). In DTD technology the only way to give occurrence constraint is to locate element more than one times in the element declaration.

*<!ELEMENT PERSONNEL ( ID, PHONE, PHONE )>*

With the declaration above, the number of ID element is set to 1 and the number of PHONE element is exactly set to 2.

**Table 6.38 Expanding number constraint**

| **Declaration** |
|---|
| <!ELEMENT CATEGORY ( CAMERA, CAMERA, CAMERA+ )> |
| **Usage** |
| <CATEGORY><br><CAMERA>...</CAMERA><br><CAMERA>...</CAMERA><br><CAMERA>...</CAMERA><br></CATEGORY> |

In table 6.38, CATEGORY element must have at least three occurrences of CAMERA elements. The number of CAMERA elements can be more than 3 but can not be less than 3.

## 6.10 Attribute List Declaration

Attributes are name-value pairs which are used in start-end and empty element tags. Between name and value = character is located. Value of an attribute is written in quotation mark. All attributes for an element are declared in the DTD unit called "attribute list" declaration (Pitts N., 2004). The information that attribute-list declaration can include is below;

- Includes the names of element's attributes. All elements' attributes must be declared here.
- Includes the types of element's attributes.
- Specifies whether or not element's attributes are required. And default values of attributes are also defined.

Attribute list declaration is defined using the keyword string <!ATTLIST in DTD mechanism. The structure of attribute list declaration is below;

*<!ATTLIST Element_Name Attribute_List>*

Declaration starts with the literal string *<!ATTLIST*, then followed *Element_Name* which specifies the name of element to which attributes are assigned. Finally declaration ends with *Attribute_List* which lists the one more attributes (Pitts N., 2004). An attribute list declaration which includes only one attribute is below;

*Attribute_Name Attribute_Type Default_Value*

Declaration starts with the name of attribute. Attribute names are unique and can not be assigned to more than one element. Attribute name also can not start with any combination of *XML* word's upper and lower letters.

**Table 6.39 Declaration of an attribute**

```
<!ATTLIST PERSONNEL ID CDATA #REQUIRED>
```

In table 6.39, PERSONNEL element has only one attribute called ID which has the data type CDATA. The default value of attribute is #REQUIRED which means that the attribute must contain a value within XML document.

**Table 6.40 Using elements and attribute declarations together**

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE PERSONNELLIST [
<!ELEMENT PERSONNELLIST (PERSONNEL+)>
<!ATTLIST PERSONNEL ID CDATA #REQUIRED>
<!ELEMENT PERSONNEL (NAME, EMAIL, (PHONE | FAX))>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
<!ELEMENT PHONE (#PCDATA)>
<!ELEMENT FAX (#PCDATA)>
```

**Table 6.40 Continued...**

```
]>

<PERSONNELLIST>
 <PERSONNEL ID="0001">
  <NAME>SIBEL KOPARAN</NAME>
  <EMAIL>SIBELK@EGETIP.EDU.TR</EMAIL>
  <PHONE>0543898709</PHONE>
 </PERSONNEL>
 <PERSONNEL ID="0002">
  <NAME> Ahmet Tan</NAM>
  <EMAIL>Ahmet.Tan@NETSIS.COM.TR</EMAIL>
  <PHONE>05328450709</PHONE>
 </PERSONNEL>
</PERSONNELLIST>
```

In table 6.40, an internal DTD in which element and attributes are defined and an XML document is showed. The XML document that contains personnel information starts with the root element PERSONNELLIST. Root element must have at least one PERSONNEL element. PERSONNEL element is the form of an ID attribute, followed NAME and EMAIL element and followed one of the PHONE or FAX elements.

**Table 6.41 Defining more than one attribute**

| **1. Usage** |
|---|
| <!ATTLIST CAMERA BRAND CDATA #REQUIRED<br>MODEL CDATA #REQUIRED> |
| **2. Usage** |
| <!ATTLIST CAMERA BRAND CDATA #REQUIRED><br><!ATTLIST CAMERA MODEL CDATA #REQUIRED> |

In table 6.41, according to PERSONNEL element declaration it has two attributes. Declaration of more than one attribute can define in two ways. Technically these two declarations are same. But the first technique decreases the size of XML document.

The name of first attribute is BRAND and type of this attribute is CDATA. The default value of attribute is #REQUIRED. And the name of first attribute is MODEL and type of this attribute is CDATA. The default value of attribute is also #REQUIRED.

## 6.11 Attribute Default Declarations

Attribute values specify that whether or not attributes are required, and if attributes are not required the fixed values are defined. 4 types of default declaration can be assigned to element attributes;

**Table 6.42 Attribute default declarations (WEB_12, 2004)**

| Declaration | Default Value | Description |
|---|---|---|
| #IMPLIED | A default value can be assigned to this value. | These types of attributes may not have an obligation to be used in XML document.<br><br>*<!ATTLIST Personnel name CDATA #IMPLIED>* |
| #REQUIRED | A default value can not be assigned to this value. | If attribute default is defined as #REQUIRED, declared element must have this attribute value within XML document. If this attribute is not used, XML document will not be valid.<br><br>*<!ATTLIST Personnel id ID #REQUIRED>* |
| #FIXED | A default value must be assigned to this value. | The attribute types which have #FIXED attribute default are constant and fixed within XML documents. Their values can not be changed in documents. If this attribute is not used, XML document will be generate the default value automatically. But if a value different from default value is used, XML document will not be valid.<br><br>*<!ATTLIST LangCode CDATA #FIXED "TR">* |
|  | A default value is assigned to this value. | If attribute default is defined using this type with no keyword, the declared attribute may not be used in within XML document. If |

| | | |
|---|---|---|
| | | this attribute is not used, XML document will be generate the default value automatically.<br><br>*<!ATTLIST List*<br>*tipi (ordered \| unordered)*<br>*"ordered">* |

## 6.12 Attribute Types

After attribute names are defined, the attribute types are declared that will constraint the attribute value. They can be examined in three categories;

- *StringType*, is the base type. This type of attributes can be any character except some exceptional characters.
- *TokenizedType*, brings many constraints to attribute values.
- *EnumeratedType*, provides to assign some value lists to attribute values.

**Table 6.43 Attribute types and categories (WEB_12, 2004)**

| Category | Attribute Type | Description |
|---|---|---|
| Character Data Type | CDATA | Includes character data. |
| Tokenized Type | ENTITY | Includes an unparsed entity name. |
| Tokenized Type | ENTITIES | Includes more than one unparsed entity names. |
| Tokenized Type | ID | Includes a unique XML name declaration. |
| Tokenized Type | IDREF | Includes a reference to a unique XML name declaration. |
| Tokenized Type | IDREFS | Includes a reference list to a unique XML name declaration. |
| Tokenized Type | NMTOKEN | Includes an XML name token. |
| Tokenized Type | NMTOKENS | Includes an XML name token list. |
| Numbered Type | NOTATION | Includes the name of notation which is declared in DTD. |
| Numbered Type | ENUMERATION | Includes the value list of attributes in DTD. |

### 6.12.1 Character Data Type(CDATA)

Character data types are base data types. This type of attributes can be any character except some exceptional characters which are "&" and "<". These characters are reserved for XML markup and called markup data. Instead of using these characters, character references (&amp; and &lt;) which carry out mapping process must be used (WEB_35, 2004). If a double or single quote is wanted to use, they must be the opposite character of the bound quote characters.

Character data type is declared using CDATA keyword string literal. Character data types can be used one of the attribute defaults #IMPLIED, #FIXED and #REQUIRED.

**Table 6.44 Valid character data type declaration**

| |
|---|
| 1. <!ATTLIST CAMERA BRAND CDATA "DEFAULTBRAND"> |
| 2. <!ATTLIST CAMERA BRAND CDATA #IMPLIED> |
| 3. <!ATTLIST CAMERA MODEL CDATA #REQUIRED> |
| 4. <!ATTLIST CAMERA TYPE CDATA #FIXED "FIXEDTYPE"> |

In table 6.44, the first BRAND attribute of CAMERA element has the default value *DEFAULTBRAND*. Although CAMERA element is used without BRAND attribute in XML document, XML parser realizes the default attribute value. 2nd BRAND attribute can not have an attribute value. In the 3rd sample, MODEL attribute must be used within CAMERA element in the XML document. In the 4th sample, TYPE attribute must have attribute type "FIXEDTYPE".

**Table 6.45 Non-valid character data types**

| |
|---|
| 1. <!ATTLIST CAMERA BRAND CDATA #IMPLIED "DEFAULTBRAND"> |
| 2. <!ATTLIST CAMERA MODEL CDATA #REQUIRED "FIXEDTYPE"> |
| 3. <!ATTLIST CAMERA TYPE CDATA #FIXED> |

In table 6.45, the reason of first and second attributes are not valid, is that they have default values. The third attribute is not valid, although it is declared with #FIXED keyword, it has a default value.

### 6.12.2 Named Token Data Type (NMTOKEN)

"Named token data types can contain attribute values; a-z and A-Z range letters, 0-9 range digits, -, _ and . characters, token characters, combination characters" (WEB_12, 2004). Named tokens can not include white-space characters but can start with alphabetical, numerical and tokenized characters.

**Table 6.46 Valid and non-valid named tokens**

| Valid named tokens | Non-valid named tokens |
|---|---|
| Book | Book 1 |
| Book1 | Book,1 |
| _Book.1 | Book! |
| _1.Book | (Book) |
| :Book | #Book |
| ____ | Book$1 |
| -Book | |
| 1.Book | |

Named token data types can be used one of the attribute defaults #IMPLIED, #FIXED and #REQUIRED like character data.

**Table 6.47 Valid named token declarations**

| |
|---|
| 1. <!ATTLIST KAMERA BRAND NMTOKEN "FIXEDBRAND"> |
| 2. <!ATTLIST KAMERA BRAND NMTOKEN #IMPLIED> |
| 3. <!ATTLIST KAMERA MODEL NMTOKEN #REQUIRED> |
| 4. <!ATTLIST KAMERA TYPE NMTOKEN #FIXED "FIXTYPE"> |

In table 6.47 all attribute declarations are valid, since all declarations are well-formed and match validity constraints.

### 6.12.3 Named Token List Data Type (NMTOKENS)

Named Tokens List Data Type includes more than one NMTOKEN in a list. List elements are separated with white spaces within double quotes (WEB_12, 2004).

**Table 6.48 Valid named token list declaration and usage**

| **Declaration** |
|---|
| <!ELEMENT PERSONNEL (#PCDATA)> <!ATTLIST PERSONNEL PHONE NMTOKENS #REQUIRED> |
| **Usage** |
| <PERSONNEL PHONE="4567438 0532783423">Sibel KOPARAN</PERSONNEL> |

### 6.12.4 ID Data Type

ID attribute type is used to distinguish XML elements to which it is assigned to. Technically, it seems tables' primary keys in relational database management systems. There can not be two XML elements which have the same ID attribute value in XML documents (WEB_12, 2004). The attribute's name may have not to be ID which has the attribute type

In real life XML applications, numeric values are wanted to be used for ID attribute types. But, Since ID attribute values which start with numeric values are not valid, "_" or alphabetical characters are added as the first character of the values.

**Table 6.49 Valid and non-valid ID attribute types**

| **Declaration** |
|---|
| <!ATTLIST PERSONNEL ID ID #REQUIRED> |
| **Non-valid attribute value** |
| <PERSONNEL ID="0001">Ahmet Tan</PERSONNEL> |
| **Valid attribute value** |
| <PERSONNEL ID="_0001">Ahmet Tan</PERSONNEL> |

In table 6.49, the attribute value which starts with numeric value is not valid. If the value of attribute is exactly wanted to be constructed using numeric values, an underscore character is added as the first character like in the second example.

**Table 6.50 Non-valid ID attribute type declarations**

| |
|---|
| 1. <!ATTLIST PERSONNEL ID ID #REQUIRED "s001"> |
| 2. <!ATTLIST PERSONNEL ID ID #IMPLIED "s001"> |
| 3. <!ATTLIST PERSONNEL ID ID #FIXED "s001"> |
| 4. <!ATTLIST PERSONNEL ID ID "s001"> |

In table 6.50, all ID type attribute declarations are non-valid. In the first one although default type is set to #REQUIRED, "s001" default attribute value is used, which is prohibited. In the second one although default type is set to #IMPLIED and has default value like in the first one. Third and fourth non-valid declarations' problems are same as with first and second one.

**Table 6.51 Valid ID attribute type declaration and usage**

| **Declaration** |
|---|
| <!ELEMENT PERSONNEL (#PCDATA)><br><!ATTLIST PERSONNEL ID ID #REQUIRED> |
| **Usage** |
| <PERSONNEL ID="s001">Ahmet Tan</PERSONNEL> |

### 6.12.5 Reference To ID Data Types (IDREF)

In real life applications it is wanted give references to XML elements which has attributes with ID types. This situation seems referential keys in relational database management systems. IDREF and IDERFS literals are used in DTDs (WEB_12, 2004).

Think a system which manages stock master information and stock transaction information. Transaction information for stocks can be stock entries or stock exits. For example, to make out an invoice is an out is a stock exit process because stock is sold.

To producing a new stock is an entry process. In this system, a reference is wanted to give to stock master information probably to a stock number.

**Table 6.52 Stock Module XML Design ID and IDREF usage**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<!DOCTYPE STOCK_MODULE [
<!ELEMENT STOCK_MODULE
(STOCK_MASTER_INFORMATION,TRANS_OPERATION,STOCK_TRANSA
CTIONS)>
<!ELEMENT STOCK_MASTER_INFORMATION (STOCK+)>
<!ELEMENT STOCK
(SNAME,SAIL_PRICE,BUY_PRICE,VAT_RATIO,GROUP_CODE)>
<!ELEMENT SNAME (#PCDATA)>
<!ELEMENT SAIL_PRICE (#PCDATA)>
<!ELEMENT BUY_PRICE (#PCDATA)>
<!ELEMENT VAT_RATIO (#PCDATA)>
<!ELEMENT GROUP_CODE (#PCDATA)>
<!ATTLIST STOCK CODE ID #REQUIRED>
<!ELEMENT TRANS_OPERATION (OPR+)>
<!ATTLIST OPR TYPE ID #REQUIRED>
<!ELEMENT OPR (NAME)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT STOCK_TRANSACTIONS (TRANS+)>
<!ATTLIST TRANS NUM ID #REQUIRED>
<!ATTLIST TRANS REFSTOCKCODE IDREF #REQUIRED>
<!ATTLIST TRANS OPR_TYPE IDREF #REQUIRED>
<!ATTLIST TRANS TYPE CDATA #REQUIRED>
<!ELEMENT TRANS (DOCNO,EXPLN,QUANTITY)>
<!ELEMENT DOCNO (#PCDATA)>
<!ELEMENT EXPLN (#PCDATA)>
<!ELEMENT QUANTITY (#PCDATA)>
]>
<STOCK_MODULE>
  <!-- Stock Master Information -->
  <STOCK_MASTER_INFORMATION>
    <STOCK CODE="S001">
      <SNAME>VR1-ONE CHANNEL RADIO CARD</SNAME>
      <SAIL_PRICE>9000000</SAIL_PRICE>
      <BUY_PRICE>8800000</BUY_PRICE>
      <VAT_RATIO>18</VAT_RATIO>
      <GROUP_CODE>RADIO</GROUP_CODE>
    </STOCK>
```

**Table 6.52 Continued...**

```
  <STOCK CODE="S002">
    <SNAME>REMOTE CONTROL DEVICE</SNAME>
    <SAIL_PRICE>8000000</SAIL_PRICE>
    <BUY_PRICE>7000000</BUY_PRICE>
    <VAT_RATIO>15</VAT_RATIO>
    <GROUP_CODE>KUMANDA</GROUP_CODE>
  </STOCK>
 </STOCK_MASTER_INFORMATION>
 <!-- Operation TRANS Information -->
 <TRANS_OPERATION>
  <OPR TYPE="F">
    <NAME>TO MAKE OUT AN INVOICE</NAME>
  </OPR>
  <OPR TYPE="U">
    <NAME>STOCK PRODCUTION</NAME>
  </OPR>
 </TRANS_OPERATION>
 <!-- Stock  TRANS Information -->
 <STOCK_TRANSACTIONS>
  <TRANS NUM="H1" REFSTOCKCODE="S001" OPR_TYPE="U"
               TYPE="ENTRY">
    <DOCNO>00001</DOCNO>
    <EXPLN>S001 STOCK PRODUCTION</EXPLN>
    <QUANTITY>5</QUANTITY>
  </TRANS>
  <TRANS NUM="H2" REFSTOCKCODE="S002" OPR_TYPE="F"
               TYPE="EXIT">
    <DOCNO>00002</DOCNO>
    <EXPLN>S002 STOCK SAILING</EXPLN>
    <QUANTITY>2</QUANTITY>
  </TRANS>
 </STOCK_TRANSACTIONS>
</STOCK_MODULE>
```

In table 6.52, an XML design of a part of stock module is showed which is included in commercial packets. According to stock module design, stock sub parts are below.

- *Stock Master Information*, is the section which includes stock master information. Each stock has an unique identifier. This attribute is the CODE of STOCK. This value can not be same within two STOCKs. Stock master information includes name, sail_price, buy_price, vat_ratio and group code in addition to code. Two stocks exist in the example;

*<STOCK CODE="S001">*

*<STOCK CODE="S002">*

....

- *Stock Transaction Operation Types* includes operation information which constitutes stock transaction. Each operation has an unique identifier called TYPE. Except the type information, operation also contains the operation name.

  *<OPR TYPE="F"> → Invoice Operation*

  *<OPR TYPE="U"> → Production Operation*

  ...

- *Stock Transactions* includes entry and exit processes after stock sailing or invoicing operations. Each transaction has a unique identifier. This attribute is the number "NUM" of transaction. The other attributes of transaction is OPR_TYPE which is the type of operation and code of the stock REFSTOCKCODE. In addition to these attribute information, the type of transaction, document number, explanation and quantity information is kept within each stock transaction.

  *<TRANS NUM="H1" REFSTOCKCODE="S001" OPR_TYPE="U" TYPE="ENTRY">*

  *<TRANS NUM="H2" REFSTOCKCODE="S002" OPR_TYPE="F" TYPE="EXIT">*

  ...

### 6.12.6 Reference Lists to Unique Identifiers (IDREFS)

IDREFS keyword is used to give reference to unique attribute lists which is defined using ID type (WEB_12, 2004). With the assistance of IDREFS type, an XML element can be related with more than one XML elements. This relation seems (One to N Relation) in relational database management systems.

If we want to create an XML report which will display stock transactions, we need to an attribute in IDREFS type which will give reference each transaction.

**Table 6.53 IDREFS attribute type and usage**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<!DOCTYPE STOCK_MODULE [
<!ELEMENT STOCK_MODULE
(STOCK_MASTER_INFORMATION,TRANS_OPERATION,STOCK_TRANSA
CTIONS, TRANS_REPORT)>
...
<!ELEMENT STOCK_TRANSACTIONS (TRANS+)>
<!ELEMENT TRANS (DOCNO,EXPLN,QUANTITY)>
...
<!ELEMENT TRANS_REPORT (REPORT)>
<!ELEMENT REPORT (#PCDATA)>
<!ATTLIST REPORT REFLIST IDREFS #REQUIRED>
]>
<STOCK_MODULE>
 <STOCK_MASTER_INFORMATION>
  ...
 </STOCK_MASTER_INFORMATION>
 <TRANS_OPERATION>
  ...
 </TRANS_OPERATION>
 <STOCK_TRANSACTIONS>
   <TRANS NUM="H1" REFSTOCKCODE="S001" OPR_TYPE="U"
TYPE="ENTRY">
   ...</TRANS>
   <TRANS NUM="H2" REFSTOCKCODE="S002" OPR_TYPE="F"
TYPE="EXIT">
    ... </TRANS>
 </STOCK_TRANSACTIONS>

 <TRANS_REPORT>
   <RAPOR REFLIST="H1 H2"> Entry Exit Transactions </RAPOR>
 </TRANS_REPORT>
</STOCK_MODULE>
```

In table 6.53, TRANS_REPORT element is added to stock module which displays stock operation transactions. This element is constituted from an REPORT sub element. To give reference to stock transactions REFLIST attribute is defined which is in the type of IDREFS. According to sample REFLIST attribute includes H1 and H2 values which means the transactions will be reported with these numbers.

### 6.12.7 Enumerated Attribute Type

If you want to assign an exact attribute value from a value list, you must define the attribute type as enumerated type. It is not necessary to use CDATA, NMTOKEN and ID like literal keywords to define enumerated types. All values are located within parenthesis by putting logical OR character "|" between them (WEB_36, 2004).

**Table 6.54 ENUMERATION attribute type declaration and usage**

| **Declaration** |
|---|
| <!ELEMENT OPERATION (#PCDATA)> |
| **Usage** |
| <!ATTLIST OPERATION TYPE (Invoice \| Account \| Production \| Check \| Local) "Local"> |

In table 6.54, a TYPE attribute which is enumerated type is defined for OPERATION element. One of the Invoice, Account, Production, Check and Local values is assigned to TYPE attribute. If nothing is assigned to TYPE attribute of OPERATION element, the default value Local is assigned.

One of the #REQUIRED or #IMPLIED types can be used as default types. But these types can not take fixed default values. Values in the list must match the rules of named tokens (NMTOKEN).

### 6.12.8 ENTITY Attribute Type

Attributes which are in ENTITY types, include the name of unparsed entities that are referenced. These attributes provide to define and reference audio, mpeg like binary and unparsed files (WEB_12, 2004).

**Table 6.55 ENTITY attribute type declaration**

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE RESOURCE ENTITY #REQUIRED>
```

In table 6.55, RESOURCE attribute for IMAGE element is defined. RESOURCE attribute is a type of ENTITY and must take the entity name as the attribute value which is located in DTD file.

**Table 6.56 ENTITY attribute type usage**

```
<IMAGE RESOURCE="IMAGEREFERENCE"/>
```

In table 6.56, IMAGE element and RESOURCE attribute usage in an XML document is showed. IMAGEREFERENCE attribute value is the name of entity that is declared in DTD mechanism.

### 6.12.9 ENTITIES Attribute Type

ENTITIES attribute type takes more than one ENTITY reference as attribute value. Reference names are located within double single quotes between commas (WEB_12, 2004).

**Table 6.57 ENTITIES attribute type and usage**

| **Declaration** |
|---|
| <!ELEMENT IMAGE EMPTY><br><!ATTLIST IMAGE RESOURCELIST ENTITIES #REQUIRED> |
| **Usage** |
| < IMAGE RESOURCELIST="RESOURCEREF1 RESOURCEREF2" /> |

In table 6.57, RESOURCELIST attribute is defined which belongs to IMAGE element and has the type of ENTITIES. RESOURCELIST attribute must contain more than one entity name. RESOURCEREF1 and RESOURCEREF2 attribute types are entity references declared in DTD.

## 6.13 Entity Declarations

Entities are special XML units which provide to define some internal and external unparsed content like image, video and audio formatted files. For example; since XSL documents are actually inherited from XML documents, they can be thought as entities (WEB_12, 2004).

- **General Entities** reference to string literals or data types within XML documents (WEB_12, 2004).
- **Parameter Entities** include string literals or text declarations in DTD files (WEB_12, 2004).
- **Internal Entities** are string literals within single or double quotes and located in the same place within XML document (WEB_12, 2004).
- **External Entities** are located in different physical files. These types of entities are referenced using URN and URI. Reference to entity content exists in XML document. <IMAGE> element in HTML markup language provides rendering of image files (WEB_12, 2004).
- **Parsed Entities** are in the form of character or markup data and are replaced with their real content during parsing processing. While content is replacing, it is controlled by parser. Content of parsed entities must be well formed XML text (WEB_12, 2004).
- **Unparsed Entities** can be character either in the range of valid XML boundaries or non-valid XML boundaries. Non-valid characters contained in image, audio or mpeg like binary files. Unparsed entity names can be assigned to attributes in the type of ENTITY or ENTITIES. Most of the XML parsers do not support usage of unparsed entities (WEB_12, 2004).

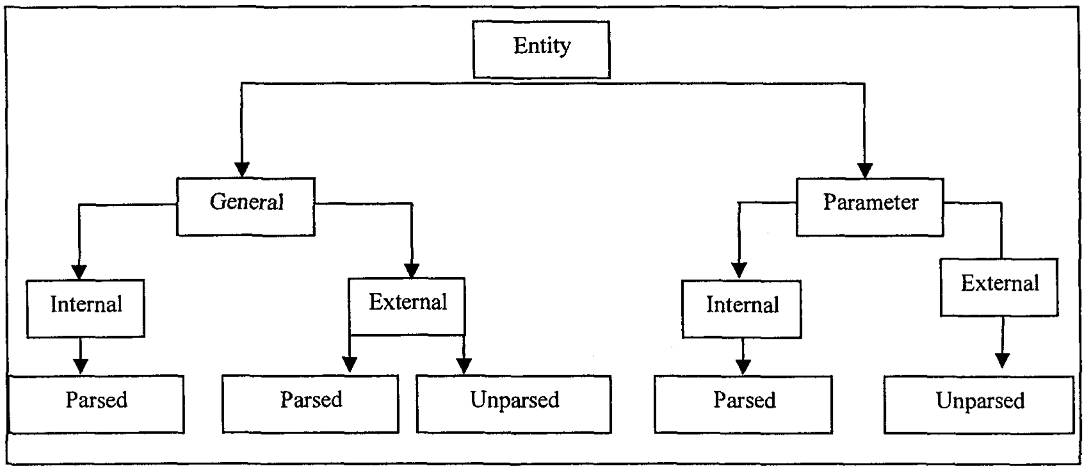


**Figure 6.1 Hierarchical structure of entity types in DTD files**

**Table 6.58 Types of entities (WEB_12, 2004)**

| Category | Type | Description |
|---|---|---|
| General | Internal Parsed | Special XML units. These entities are validated during replacement process. And after controlling, they are replaced by their real content. |
| | External Parsed | Located in different physical files. These entities are validated during replacement process and they are replaced by their real content. |
| | External Unparsed | Located in different physical files and are not validated during replacement process. Content of these entities are image, video audio like binary files. |
| Parameter | Internal Parsed | They provide to define DTD units like ATTLIST, ELEMENT. They can be thought as shortcuts for DTD unit declarations. |
| | External Parsed | Technically same as Internal Parsed Entities. But they are located different location from DTD. |

### 6.13.1 General Internal Parsed Entities

General internal parsed entities (WEB_12, 2004) are special XML units. These entities are validated during replacement process. And after controlling, they are replaced by their real content. Content of parsed entities must be well-formed XML text.

They are declared using ENTITY keyword in the DTD file. "&" and ";" characters are used to reach internal entities in XML documents.

**Table 6.59 Internal parsed entity declaration**

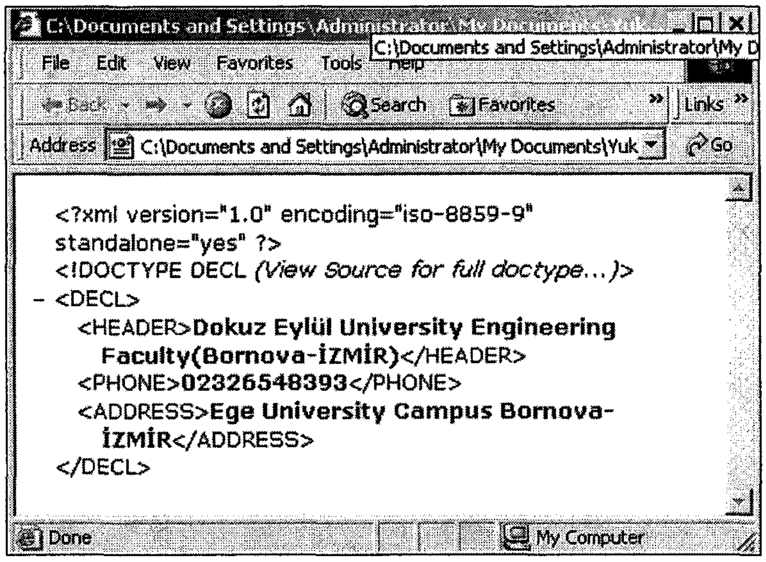
```
<!ENTITY deuEF "Dokuz Eylül University Engineering Faculty">
```

In table 6.59, *deuEF* entity is like a shortcut for the *Dokuz Eylül University Engineering Faculty* content. Instead of writing this lengthy content, simple *&deuEF;* is used.

**Table 6.60 Internal parsed entity definition and usage**

```
<?xml version="1.0" encoding="iso-8859-9" standalone="yes" ?>
<!DOCTYPE DECL[
  <!ENTITY deuEF "Dokuz Eylül University Engineering Faculty">
  <!ENTITY engPlace "Bornova-İZMİR">
  <!ELEMENT DECL (HEADER,PHONE,ADDRESS)>
  <!ELEMENT HEADER (#PCDATA)>
  <!ELEMENT PHONE (#PCDATA)>
  <!ELEMENT ADDRESS (#PCDATA)>
]>
<DECL>
  <HEADER>&deuEF;(&engPlace;)</HEADER>
  <PHONE>02326548393</PHONE>
  <ADDRESS>Ege University Campus &engPlace;</ADDRESS>
</DECL>
```

In table 6.60, *deuEF* and *engPlace* entities' declaration and usage within XML document are showed. engPlace entity is use both in ADDRESS element's content and HEADER element's content. Although HEADER AND ADDRESS elements are defined in PCDATA type, they include & character, since that is the start character of entity.


C:\Documents and Settings\Administrator\My Documents\Yuk
File Edit View Favorites Tools Help
Back Search Favorites Links
Address C:\Documents and Settings\Administrator\My Documents\Yuk
Go
Done
My Computer

```
<?xml version="1.0" encoding="iso-8859-9"
standalone="yes" ?>
<!DOCTYPE DECL (View Source for full doctype...)>
- <DECL>
    <HEADER>Dokuz Eylül University Engineering
      Faculty(Bornova-İZMİR)</HEADER>
    <PHONE>02326548393</PHONE>
    <ADDRESS>Ege University Campus Bornova-
      İZMİR</ADDRESS>
  </DECL>
```

**Figure 6.2 Rendering of table 5.60 using Internet Explorer**

In figure 6.2, web representation of table 6.60 is showed using Internet Explorer browser. Declaring frequently used text in the general entity form is very useful. When content will be changed, it will not be necessary to find and replace the all used literals, only DTD declaration is changed.

If an XML document has both internal definition and external definition and has the entities or attribute list which have the same names, internal definitions are always more priority then external.

**Table 6.61 Using internal and external declarations together**

| **Declaration** |
|---|
| <?xml version="1.0" encoding="iso-8859-9"?> |
| <!DOCTYPE SCHOOL SYSTEM "ext.DTD" [ |
| <!ELEMENT SCHOOL (#PCDATA)> |
| <!ENTITY schoolname "DEU"> |
| ]> |
| <SCHOOL>&schoolname;</SCHOOL> |
| **EXT.DTD** |
| <!ENTITY schoolname "Dokuz Eylul University"> |

In table 6.61, schoolname is defined both in internal DTD and in external EXT.DTD. Since internal DTD has more precedence, SCHOOL element will have the "DEU" content.

### 6.13.2 Predefined General Entities

In XML technology 5 predefined entity references are defined (WEB_12, 2004). It is not needed to define these entities in DTD again.

**Table 6.62 Predefined entities (WEB_12, 2004)**

| Usage | Declaration | Description |
|---|---|---|
| &amp;amp; | &lt;!ENTITY amp "&amp;#38;#38;"&gt; | & (and) character. Used as the first character of entity references. |
| &amp;lt; | &lt;!ENTITY lt "&amp;#38;#60;"&gt; | &lt; character. It is the starting character of XML element's tags. |
| &amp;gt; | &lt;!ENTITY gt "&amp;#62;"&gt; | &gt; character. It is the closing character of XML element's tags. |
| &amp;quot; | &lt;!ENTITY quot "&amp;#34;"&gt; | "(quote) character. Used for to keep XML attribute values. |
| &amp;apos; | &lt;!ENTITY apos "&amp;#39;"&gt; | ' (apos) character. Used for to keep XML attribute values. |

### 6.13.3 General Internal Entity References Usage Situations

- It is possible to use an entity reference in entity definition.

**Table 6.63 Using entity references in entity declarations**

```
<!ENTITY deuEF "Dokuz Eylül University Engineering Faculty">
<!ENTITY engPlace "&deuEF; Bornova-İZMİR">
```

- All characters except %, & and " can be used in entity declarations. If one of these characters must be used, character references which are mapped then are chosen (WEB_12, 2004).

**Table 6.64 Using markup as the content of entities**

```
<!ENTITY  declContent
   "<HEADER>&deuEF;(&engPlace;)</HEADER>
    <PHONE>02326548393</PHONE>
    <ADDRESS>Ege University Campus &engPlace;</ADDRESS>">
```

- Entities can not be used with cross dependencies.

**Table 6.65 Cross dependent entity declarations**

```
 <!ENTITY deuEF "Dokuz Eylül University Eng. Faculty
&engPlace;">
 <!ENTITY engPlace "&deuEF; Bornova-İZMİR">
```

- Entity references can not be use within DTD element type definitions.

**Table 6.66 Entity usage in element definition**

```
<!ENTITY deuEF "#PCDATA">
<!ELEMENT HEADER &deuEF;>
<!ELEMENT PHONE &deuEF;>
```

### 6.13.4 General External Parsed Entities

Located in different physical files and are validated during replacement process. Since they are validated their text content must be well-formed. General external parsed entities are defined using ENTITY keyword in DTD mechanism. After ENTITY keyword external DTDs are referenced using “SYSTEM” literal (WEB_39_2004). Finally, URI (Uniform Resource Identifier) address follows which holds the location information of content (WEB_12, 2004).

**Table 6.67 General external parsed entity definition**

```
<!ENTITY generalKey SYSTEM "http://www.netsis.com.tr/key.XML">
<!ENTITY generalKey SYSTEM "/keyList/key.XML">
```

In table 6.67 two different entity declarations is showed which are named generalKey. Two definitions are specified using different SYSTEM address. In the first definition an URL address and in the second definition an URI address is used.

**Table 6.68 External key file**

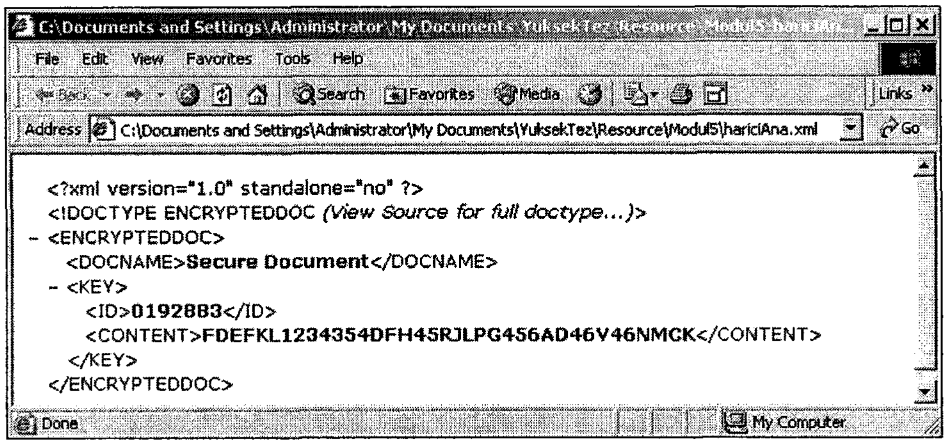
```
<?xml version="1.0"?>
<KEY>
 <ID>0192883</ID>
<CONTENT>FDEFKL1234354DFH45RJLPG456AD46V46NMCK</CONTENT>
</KEY>
```

In table 6.68, an XML key file is showed that can be used in real life security applications. The security file can be used as key content in encryption or decryption algorithms.

**Table 6.69 General external parsed entity usage**

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE ENCRYPTEDDOC[
  <!ELEMENT ENCRYPTEDDOC (DOCNAME,KEY)>
  <!ELEMENT KEY (ID,CONTENT)>
  <!ELEMENT ID (#PCDATA)>
  <!ELEMENT CONTENT (#PCDATA)>
  <!ELEMENT DOCNAME (#PCDATA)>
  <!ENTITY enKEY SYSTEM "externalKEY.xml">
]>
<ENCRYPTEDDOC>
  <DOCNAME>Secure Document</DOCNAME>
  &enKEY;
</ENCRYPTEDDOC>
```

In table 6.69, an entity named “enKEY” is defined which references an external XML file *externalKEY.xml.* By using *&enKEY;* entity after DOCNAME element, external files content is added to XML document. If it is paid attention, DTD definition of external XML file is declared in the internal one.


C:\Documents and Settings\Administrator\My Documents\YuksekTez\Resource\Modul5\hariciAna.xml
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Documents and Settings\Administrator\My Documents\YuksekTez\Resource\Modul5\hariciAna.xml
Go Links
Done
My Computer

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE ENCRYPTEDDOC (View Source for full doctype...)>
- <ENCRYPTEDDOC>
    <DOCNAME>Secure Document</DOCNAME>
  - <KEY>
      <ID>0192883</ID>
      <CONTENT>FDEFKL1234354DFH45RJLPG456AD46V46NMCK</CONTENT>
    </KEY>
  </ENCRYPTEDDOC>
```

**Figure 6.3 Web rendering of external parsed entities**

### 6.13.5 General External Entity References Usage Situations

- It is possible to use another entity reference in the definition of general external declaration.
- If entity declaration wholly contains markup data, data must be well-formed. Starting and closing tags must match each other and well-formedness constraints.
- Since Internal XML document has already a root element, it is not necessary that entity XML content must contain a root element.
- Single or double quotes used in entities' contents must be different from boundary characters.
- Cross dependency is forbidden.

### 6.13.6 General External Unparsed Entities

Located in different physical files but they are not validated during replacement process like general external parsed entities. They can not be in the form of XML text. XML parser does not control these entities' content and pass them directly to application (WEB_12, 2004). Shortly, XML parser believe content's reliability.

External unparsed entities are generally used when GIF, JPEG, WAV and AVI like binary files are wanted to attach XML documents (WEB_39, 2004). For example, a stock's image can be demanded in the XML file or an XML document which contains films' information may contain film previews.

External unparsed entities can not be specified using entity references. That is to say a declaration like *&entity_name;* can not be done (WEB_12, 2004). Attributes in the type of ENTITY or ENTITIES literals must be used to reach unparsed entity resources.

**Table 6.70 ENTITY type usage**

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE SOURCE ENTITY #REQUIRED>
```

In table 6.70, an empty IMAGE element and its SOURCE attribute is defined. Attributes which are in ENTITY types can be used with #IMPLIED or #FIXED attribute defaults. To able to use general external entities, two definitions must be created in DTD mechanism.

- Notation Declarations
- External Unparsed Entity Definitions

### 6.13.7 Notation Declarations

"Notations identify by name the format of unparsed entities, the format of elements" (WEB_12, 2004). Notations use URL and MIME types to identify content environment. One method of rendering of these types of content is creating fix markup names. For example <IMG> tag in HTML is used represent GIF, JPEG, PNG and BMP like image files. Approach for fixing tags obstructs to define our tag set. Somewhat apart understanding file type from its format may not be a smart method.

Notations bring a good solution to specify the content type of non-XML data. Notations are defined with the same level elements and attributes using NOTATION literal string in DTD mechanism. Each notation definition has a notation name and an external identifier. A typical notation declaration is below;

<!NOTATION *notation_name* SYSTEM *"external_identifier"*>

(WEB_12, 2004)

- Notation name is the name of format information which specifies content types in DTD.
- External identifiers are used to specify content format type. There exist three ways to define external identifiers;
    - **MIME types** can be used;
      *<!NOTATION GIF SYSTEM "image/gif">*
    - **PUBLIC identifiers** can be used instead of SYSTEM identifiers since SYSTEM identifiers may be change frequently (WEB_37, 2004).
      *<!NOTATION GIF PUBLIC*
      *"-//IETF//NONSGML Media Type image/gif//WIDTH "*
      *"http://www.isi.edu/in-notes/iana/assignments/media-types/image/gif">*
    - **ISO and IETF** like universal standards can be used; For example ISO 8601 standard specifies date and time format.

### 6.13.8 External Unparsed Entity Definitions

They provide to reach to real content path of unparsed content (WEB_12, 2004). A typical external unparsed entity definition is below;

<!ENTITY *entityName* SYSTEM contentInfo *NDATA notationName*>

- *entityName* is the name unparsed entity. Entity name must match the XML naming rules.

- *contentInfo* is the content path of URI which is mapped to unparsed entity's real content.
- *NDATA* string literal shows that external entity contains unparsed content.
- *notationName* is the location of program which specifies entity content information.

**Table 6.71 External Unparsed Entity Declarations**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<!DOCTYPE STOCK_MASTER_INFORMATION  [
<!ELEMENT STOCK_MASTER_INFORMATION (STOCK+)>
<!ELEMENT STOCK (NAME,PRICE,IMAGE)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ATTLIST STOCK CODE ID #REQUIRED>
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE PATH ENTITY #REQUIRED>
<!NOTATION GIF SYSTEM "image/gif ">
<!ENTITY s01Res SYSTEM "S01.gif" NDATA GIF>
<!ENTITY s01Res SYSTEM "S02.gif" NDATA GIF>
]>

<STOCK_MASTER_INFORMATION>
 <STOCK CODE="S001">
  <NAME>VR1-ONE CHANNEL RADIO CARD </NAME>
  <PRICE>1.000.000</PRICE>
  <IMAGE PATH="s01Res"/>
 </STOCK>
 <STOCK CODE="S002">
  <NAME>TO3- THREE CHANNELS REMOTE CONTROL
            DEVICE</NAME>
  <PRICE>1.500.000</PRICE>
  <IMAGE PATH="s02Res"/>
 </STOCK>
</STOCK_MASTER_INFORMATION>
```

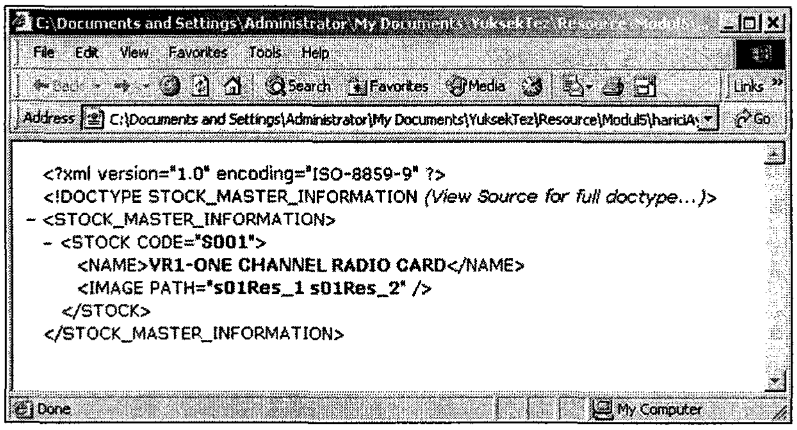
In table 6.71 an XML document which contains company stock information and DTD declaration which constraints XML. An IMAGE element which is in the type of ENTITY is defined to keep stock images. GIF notation declarations specifies format of images. PATH attribute of IMAGE element may include *s01Res* or *s02Res* entities as attribute value which maps *S01.gif* and *S02.gif* image files.

If it is wanted to reference more than one entity content from an element attribute, ENTITIES literal must be used. Entities are lined up between spaces in the attribute value.

**Table 6.72 Referencing to external unparsed entities**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<!DOCTYPE STOCK_MASTER_INFORMATION  [
<!ELEMENT STOCK_MASTER_INFORMATION (STOCK+)>
<!ELEMENT STOCK (NAME,IMAGE)>
<!ELEMENT NAME (#PCDATA)>
<!ATTLIST STOCK CODE ID #REQUIRED>
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE PATH ENTITIES #REQUIRED>
<!NOTATION GIF SYSTEM "image/gif ">
<!ENTITY s01Res_1 SYSTEM "S01_1.gif" NDATA GIF>
<!ENTITY s01Res_2 SYSTEM "S01_2.gif" NDATA GIF>
]>
<STOCK_MASTER_INFORMATION>
  <STOCK CODE="S001">
    <NAME> VR1-ONE CHANNEL RADIO CARD </NAME>
    <IMAGE PATH=" s01Res_1 s01Res_2"/>
  </STOCK>
</STOCK_MASTER_INFORMATION>
```

In table 6.72 an IMAGE element in the type of *ENTITIES* instead of ENTITY is defined to keep company stocks which may have more than one image resource. For example one of the stock's images can be small and another one can be in larger sizes. It is not to be forgotten that unparsed entity content can not be rendered by web browsers directly without using CSS files. These entities are passed to application which processes XML document and are used by application.


C:\Documents and Settings\Administrator\My Documents\YuksekTez\Resource\Modul5...
File Edit View Favorites Tools Help
Back Search Favorites Media Links
Address C:\Documents and Settings\Administrator\My Documents\YuksekTez\Resource\Modul5\hariciA\ Go
Done My Computer

```
<?xml version="1.0" encoding="ISO-8859-9" ?>
<!DOCTYPE STOCK_MASTER_INFORMATION (View Source for full doctype...)>
- <STOCK_MASTER_INFORMATION>
  - <STOCK CODE="S001">
      <NAME>VR1-ONE CHANNEL RADIO CARD</NAME>
      <IMAGE PATH="s01Res_1 s01Res_2" />
    </STOCK>
</STOCK_MASTER_INFORMATION>
```

**Figure 6.4 Rendering unparsed entities using browsers**

### 6.13.9 Internal Parsed Parameter Entities

Although general entities can be defined in DTDs, they are not a piece of DTD file and they belong to XML document body (WEB_39, 2004). They provide to define DTD units like ATTLIST, ELEMENT. They can be thought as shortcuts for DTD unit declarations. They can not be used in except DTD definition in XML document. Parameter entity references are located between "%" and ";" characters.

**Table 6.73 Internal parsed parameter entity definition**

```
<!ENTITY % deuEF "Dokuz Eylül University Engineering Faculty">
```

In table 6.73 a parameter entity definition declaration with the name of deuEF is showed. Technically they seemed to general entity definitions. The only difference that before parameter entity name "%" character is used.

**Table 6.74 Dependent parameter entity definitions**

```
<!ENTITY % engPlace "Bornova-İZMİR">
<!ENTITY % deuEF "Dokuz Eylül University Engineering Faculty %engPlace;">
```

In table 6.74, it is possible to use another parameter entity reference in a parameter entity definition. Parameter entities prevent to define most frequently used DTD elements and attributes. When content will be changed, it will not be necessary to find and replace the all used literals, only DTD declaration is changed.

**Table 6.75 Internal parsed parameter entity definitions**

| **DTD Definition 1** |
|---|
| <!ELEMENT FRUITLIST (APPLE,PEAR,BANANA, CHERRY)> |
| <!ELEMENT APPLE (PRODUCTION_LOC, PRICE)> |
| <!ELEMENT PEAR (PRODUCTION_LOC, PRICE)> |
| <!ELEMENT BANANA (PRODUCTION_LOC, PRICE)> |
| <!ELEMENT CHERRY (PRODUCTION_LOC, PRICE)> |
| <!ELEMENT PRODUCTION_LOC (#PCDATA)> |
| <!ELEMENT PRICE (#PCDATA)> |
| **DTD Definition 2** |
| <!ELEMENT FRUITLIST (APPLE,PEAR,BANANA, CHERRY)> |
| <!ENTITY % **fruit_sublist** "(PRODUCTION_LOC, PRICE)"> |
| <!ENTITY % **sublist_type** "#PCDATA"> |
| <!ELEMENT APPLE  %fruit_sublist;> |
| <!ELEMENT PEAR %fruit_sublist;> |
| <!ELEMENT BANANA %fruit_sublist;> |
| <!ELEMENT CHERRY %fruit_sublist;> |
| <!ELEMENT PRODUCTION_LOC %sublist_type;> |
| <!ELEMENT PRICE %sublist_type;> |

In table 6.75 a DTD definition is showed which contains DTD definitions. According to definition each fruit element has PRODUCTION_LOC and PRICE sub elements (child elements) and all of the child elements are in the form of parsed character data #PCDATA. If it is wanted to add a new child element to fruit elements, all of the fruit elements would have to be updated one by one and that's time consuming. With the usage of parameter entities *fruit_sublist* and *sublist_type* this problem is solved since from it will be enough to change only these parameters.

**Table 6.76 Parameter entity usage in element definition**

```
<!ENTITY deuEF "#PCDATA">
<!ELEMENT HADER &deuEF;>
<!ELEMENT PHONE &deuEF;>
```

Internal parameter entities must contain the whole element definition they can not contain a part of definition.

**Table 6.77 Valid and non-valid parameter entity usages**

```
<!DOCTYPE SCHOOL [
<!ELEMENT SCHOOL (NAME,CITY)>
<!--Since it contains whole element type definition, it is a valid parameter entity
definition -->
<!ENTITY % nameDecl "<!ELEMENT NAME (#PCDATA)>">
%nameDecl;
<!--Since it does not contain whole element type definition, it is not a valid
parameter entity definition -->
<!ENTITY % cityDecl "CITY (#PCDATA)">
<!ELEMENT %cityDecl ;>
-->
<!ELEMENT CITY (#PCDATA)>
]>
<SCHOOL>
    <NAME>Dokuz Eylül University</NAME>
    <CITY>İzmir</CITY>
</SCHOOL>
```

In table 6.77, NAME element is defined using a parameter entity reference. Since *nameDecl* reference includes all data that will be used in element declaration, it is a true definition. But, PE reference that is defined for CITY element is not fulfilled, because it does not include the characters <, >,! and the ELEMENT keyword. Usage of *cityDecl* entity like <!ELEMENT %cityDecl;> does not fit WFC. If it had all element definition, it would be valid.

### 6.13.10 External Parsed Parameter Entities

External parameter entities are defined out of the XML documents and they are like as shortcuts for DTD unit declarations (WEB_12, 2004). The difference of external parsed parameter entities from internal parsed entities is the usage SYSTEM keyword after the entity name and it is followed entity content path in URI (Uniform Resource Identifier) standard (WEB_39, 2004).

Internal parameter entities can be used only DTD mechanisms to which belong to. From another documents it is not possible to access internal parameter entities. External parameter entities provide to call a DTD from another DTD by attaching shortcuts to external DTDs. Bigger DTDs are created from smaller ones by using common external parameter entities

**Table 6.78 External parameter entity references StockMaster**

```
<!--Stock Master Information-->
<!ELEMENT STOCK (SNAME, GROUP_CODE, MEAS_UN1, LOT_1,
DENOM_1, MEAS_UN2, LOT_2, DENOM_2, MEAS_UN3,
ACC_DETAILCODE, UNIT_WEIGHT, VAT_RATIO, STORE_CODE, LOCK,
SAILLOCK, BARCODE1, BARCODE2, WIDTH, HEIGHT, WIDENESS,
ISCOMPONENT, ISPRODUCT, EXC_BUYING_PRICE,
EXC_SAILING_PRICE, SAIL_PRICE1, SAIL_PRICE2, BUY_PRICE1,
BUY_PRICE2)>
<!ATTLIST STOCK CODE ID #REQUIRED>
<!ELEMENT SNAME (#PCDATA)>
<!ELEMENT GROUP_CODE (#PCDATA)>
<!ELEMENT MEAS_UN1 (#PCDATA)>
<!ELEMENT LOT_1 (#PCDATA)>
<!ELEMENT DENOM_1 (#PCDATA)>
<!ELEMENT MEAS_UN2 (#PCDATA)>
<!ELEMENT LOT_2 (#PCDATA)>
<!ELEMENT DENOM_2 (#PCDATA)>
<!ELEMENT MEAS_UN3 (#PCDATA)>
<!--Account Detail Code-->
<!ELEMENT ACC_DETAILCODE (#PCDATA)>
<!ELEMENT UNIT_WEIGHT (#PCDATA)>
<!ELEMENT VAT_RATIO (#PCDATA)>
<!-- Store Code-->
<!ELEMENT STORE_CODE (#PCDATA)>
<!-- Locked Stocks-->
<!ELEMENT LOCK (#PCDATA)>
<!-- Sailed Locked Stocks-->
<!ELEMENT SAILLOCK (#PCDATA)>
<!ELEMENT BARCODE1 (#PCDATA)>
```

**Table 6.78 Continued...**
```
<!ELEMENT BARCODE2 (#PCDATA)>
<!ELEMENT WIDTH (#PCDATA)>
<!ELEMENT HEIGHT (#PCDATA)>
<!ELEMENT WIDENESS (#PCDATA)>
<!-- It is a sub product in the BOM (Bill of Materials) and it can not include any
sub product -->
<!ELEMENT ISCOMPONENT (#PCDATA)>
<!-- It is a product in the BOM (Bill of Materials) and it can include sub products
 -->
<!ELEMENT ISPRODUCT (#PCDATA)>
<!--Exchange Buying Price-->
<!ELEMENT EXC_BUYING_PRICE (#PCDATA)>
<!--Exchange Sailing Price-->
<!ELEMENT EXC_SAILING_PRICE (#PCDATA)>
<!ELEMENT SAIL_PRICE1 (#PCDATA)>
<!ELEMENT SAIL_PRICE2 (#PCDATA)>
<!ELEMENT BUY_PRICE1 (#PCDATA)>
<!ELEMENT BUY_PRICE2 (#PCDATA)>
```

In table 6.78 a DTD file which will be used as external parameter entity reference from another DTD. This file defined sub elements of a stock element. By using this DTD a common stock declaration is created.

**Table 6.79 External parameter entity references StockOperation**

```
<!--Stock Transactions Operation Information-->
<!ELEMENT TRANS_OPERATION (OPR+)>
<!ATTLIST OPR TYPE ID #REQUIRED>
<!ELEMENT OPR (NAME)>
<!ELEMENT NAME (#PCDATA)>
```

In table 6.79, a DTD file which will be used as external parameter entity reference from another DTD that contains stock transactions' operation information.

**Table 6.80 External parameter entity references StockTransaction**

```
<!--Stock Transaction Information-->
<!ELEMENT STOCK_TRANSACTIONS (TRANS+)>
```

**Table 6.80 Continued...**

```
<!ATTLIST TRANS NUM ID #REQUIRED>
<!ATTLIST TRANS REFSTOCKCODE IDREF #REQUIRED>
<!ATTLIST TRANS OPR_TYPE IDREF #REQUIRED>
<!ATTLIST TRANS TYPE CDATA #REQUIRED>
<!ELEMENT TRANS (DOCNO, EXPLN, QUANTITY, DATE,
NETPRICE, GROSSPRICE, TRANS_VAT, TRANS_ROWDIS,
TRANS_PROADDDIS, TRANS_FTIRSIP, TRANS_PAYDAY,
TRANS_ORDNUM, TURN, WAYBILL_NO,
TRANS_DELDATE, ACC_CODE)>
<!ELEMENT DOCNO (#PCDATA)>
<!ELEMENT EXPLN (#PCDATA)>
<!ELEMENT QUANTITY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!--Net Price-->
<!ELEMENT NETPRICE (#PCDATA)>
<!--Gross Price-->
<!ELEMENT GROSSPRICE (#PCDATA)>
<!--KDV-->
<!ELEMENT TRANS_VAT (#PCDATA)>
<!--Row Discount Sum-->
<!ELEMENT TRANS_ROWDIS  (#PCDATA)>
<!--Production Addition Discount-->
<!ELEMENT TRANS_PROADDDIS (#PCDATA)>
<!--Invoice/Waybill Type-->
<!ELEMENT TRANS_FTIRSIP (#PCDATA)>
<!--Pay Day-->
<!ELEMENT TRANS_PAYDAY  (#PCDATA)>
<!--Order No-->
<!ELEMENT TRANS_ORDNUM  (#PCDATA)>
<!ELEMENT TURN  (#PCDATA)>
<!ELEMENT WAYBILL_NO (#PCDATA)>
<!ELEMENT TRANS_DELDATE (#PCDATA)>
<!--Account Code-->
<!ELEMENT ACC_CODE (#PCDATA)>
```

In table 6.80, a DTD file which will be used as external parameter entity reference from another DTD that contains stock transaction information.

**Table 6.81 External parameter entity references StockModule**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<!DOCTYPE STOCK_MODULE  [
<!ELEMENT STOCK_MODULE
(STOCK_MASTER_INFORMATION,TRANS_OPERATION,STOCK_TRANSA
CTIONS)>
<!ENTITY % entStMaster SYSTEM "extStockMaster.dtd">
%entStMaster;
<!ENTITY % entStOprType SYSTEM "extStockOpr.dtd">
%entStOprType;
<!ENTITY % entStTrans SYSTEM "extStockTrans.dtd">
%entStTrans;
]>

<STOCK_MODULE>
 <STOCK_MASTER_INFORMATION>
  <STOCK CODE="S001">
   <SNAME>VR1-ONE CHANNEL RADIO CARD</SNAME>
   <GROUP_CODE>RADIO</GROUP_CODE>
   <MEAS_UN1>AD</MEAS_UN1>
   <LOT_1>1</LOT_1>
   <DENOM_1>12</DENOM_1>
   <MEAS_UN2>KL</MEAS_UN2>
   <LOT_2>1</LOT_2>
   <DENOM_2>36</DENOM_2>
   <MEAS_UN3>PK</MEAS_UN3>
   <ACC_DETAILCODE>102-01-01-0001</ACC_DETAILCODE>
   <UNIT_WEIGHT>20</UNIT_WEIGHT>
   <VAT_RATIO>18</VAT_RATIO>
   <STORE_CODE>1</STORE_CODE>
   <LOCK>H</LOCK>
   <SAILLOCK>H</SAILLOCK>
   <BARCODE1>S005</BARCODE1>
   <BARCODE2>S006</BARCODE2>
   <WIDTH>5</WIDTH>
   <HEIGHT>12</HEIGHT>
   <WIDENESS>3</WIDENESS>
   <ISCOMPONENT>H</ISCOMPONENT>
   <ISPRODUCT>E</ISPRODUCT>
   <EXC_BUYING_PRICE>100</EXC_BUYING_PRICE>
   <EXC_SAILING_PRICE>120</EXC_SAILING_PRICE>
   <SAIL_PRICE1>9000000</SAIL_PRICE1>
   <BUY_PRICE1>8800000</BUY_PRICE1>
   <SAIL_PRICE2>9100000</SAIL_PRICE2>
```

**Table 6.81 Continued...**

```
    <BUY_PRICE2>8900000</BUY_PRICE2>
     <VAT_RATIO>18</VAT_RATIO>
   </STOCK>
  </STOCK_MASTER_INFORMATION>
  <TRANS_OPERATION>
   <OPR TYPE="F">
     <NAME>TO MAKE OUT AN INVOICE</NAME>
   </OPR>
  </TRANS_OPERATION>
  <STOCK_TRANSACTIONS>
   <TRANS NUM="1" REFSTOCKCODE="S001" OPR_TYPE="F"
TYPE="EXIT">
     <DOCNO>00001</DOCNO>
     <EXPLN>SAILING OF S001 STOCK</EXPLN>
     <QUANTITY>2</QUANTITY>
     <DATE>29/10/2003</DATE>
     <NETPRICE>9000000</NETPRICE>
     <GROSSPRICE>9000000</GROSSPRICE>
     <TRANS_VAT>18</TRANS_VAT>
     <TRANS_ROWDIS>0</TRANS_ROWDIS>
     <TRANS_PROADDDIS>0</TRANS_PROADDDIS>
     <TRANS_FTIRSIP>1</TRANS_FTIRSIP>
     <TRANS_PAYDAY>0</TRANS_PAYDAY>
     <TRANS_ORDNUM>SIP0001</TRANS_ORDNUM>
     <TURN>1</TURN>
     <WAYBILL_NO>IRS0001</WAYBILL_NO>
     <TRANS_DELDATE>29/10/2003</TRANS_DELDATE>
     <ACC_CODE>102-01-01-0001</ACC_CODE>
   </TRANS>
  </STOCK_TRANSACTIONS>
</STOCK_MODULE>
```

In table 6.81, main stock module XML document and its DTD declaration is showed. Form this document three references are given by using external parameter entity references that are below;

```
<!ENTITY % entStMaster SYSTEM "extStockMaster.dtd">
%entStMaster;
<!ENTITY % entStOprType SYSTEM "extStockOpr.dtd">
%entStOprType;
```

```
<!ENTITY % entStTrans SYSTEM "extStockTrans.dtd">
%entStTrans;
```

These references provide to decrease the size of XML document and make easier to manage the whole domain. With the modularity references can be used from other XML documents.

## 6.14 Conditional Sections

"Conditional sections are units in document type declaration external subset which are included in, or excluded from, the logical structure of the DTD based on the keyword" (Harold E.R, 1999).

"If the keyword of the conditional section is INCLUDE, then the contents of the conditional section are part of the DTD. If the keyword of the conditional section is IGNORE, then the contents of the conditional section are not logically part of the DTD" (Harold E.R, 1999). Note that for reliable parsing, the contents of even ignored conditional sections must be read in order to detect nested conditional sections and ensure that the end of the outermost (ignored) conditional section is properly detected.

**Table 6.82 Using conditional sections**

| **Sample 1** |
|---|
| <![INCLUDE[<br>  <!ELEMENT Book (comment*,header, content, add?)><br>]]> |
| **Sample 2** |
| <![IGNORE[<br>  <!ELEMENT Book (header, content, add?)><br>]]> |
| **Sample 3** |
| <![IGNORE[<br>  <!ELEMENT Book (header, content, add?)><br>  <![INCLUDE[<br>    <!ELEMENT header (head1, head2, head3)> |

**Table 6.82 Continued...**

```
  ]]>
]]>
```

In table 6.82, definitions are showed which belong to INCLUDE and IGNORE sections. In the third definition, inner INCLUDE conditional section has no function since it is located in an IGNORE section. Conditional sections can be more productive with the parameter entities.

**Table 6.83 Using parameter entities or conditional sections**

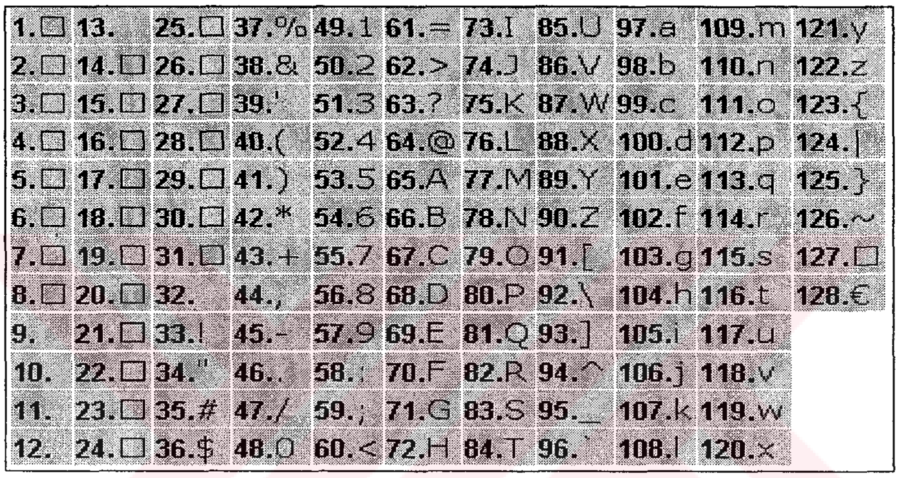
```
<!ENTITY % entIgn “IGNORE”>
<!ENTITY % entInc “INCLUDE”>
<![%entIgn; [
    <!ENTITY % entStMaster SYSTEM "extStockMaster1.dtd">
     %entStMaster;
]]>
<![%entIgn; [
    <!ENTITY % entStMaster SYSTEM "extStockMaster2.dtd">
     %entStMaster;
]]>
```

In table 6.83, *entIgn* and *entInc* entities contain the IGNORE and INCLUDE keywords which specifies conditional sections. In this manner when content will be changed, it will not be necessary to find and replace the all used literals.

## 6.15 Character Entities

XML supports both UCS-2 and Unicode standards. In addition to these standards, UTF-8 and UTF-16 (UTF, UCS Transformation Format) are also supported. “UCS and Unicode standards, in order to remain backward-compatible with existing text files, have the same first 128 characters as the 128 characters in the ASCII character set” (WEB_12, 2004). Character references can be described with two methods;

- In the first method, the characters are declared using base 10 system which are mapped to ISO/IEC 10646 codes. Character reference starts with &# literal. The string *&#65* represents upper case "A" character.
- In the second method, the characters are declared using base 16 system (hexadecimal) which are mapped to ISO/IEC 10646 codes. Character reference starts with &#x literal. The string *&#x41* represents upper case "A" character.



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1.□ | 13. | 25.□ | 37.% | 49.1 | 61.= | 73.I | 85.U | 97.a | 109.m | 121.y |
| 2.□ | 14.□ | 26.□ | 38.& | 50.2 | 62.> | 74.J | 86.V | 98.b | 110.n | 122.z |
| 3.□ | 15.□ | 27.□ | 39.' | 51.3 | 63.? | 75.K | 87.W | 99.c | 111.o | 123.{ |
| 4.□ | 16.□ | 28.□ | 40.( | 52.4 | 64.@ | 76.L | 88.X | 100.d | 112.p | 124.\| |
| 5.□ | 17.□ | 29.□ | 41.) | 53.5 | 65.A | 77.M | 89.Y | 101.e | 113.q | 125.} |
| 6.□ | 18.□ | 30.□ | 42.* | 54.6 | 66.B | 78.N | 90.Z | 102.f | 114.r | 126.~ |
| 7.□ | 19.□ | 31.□ | 43.+ | 55.7 | 67.C | 79.O | 91.[ | 103.g | 115.s | 127.□ |
| 8.□ | 20.□ | 32. | 44., | 56.8 | 68.D | 80.P | 92.\ | 104.h | 116.t | 128.€ |
| 9. | 21.□ | 33.! | 45.- | 57.9 | 69.E | 81.Q | 93.] | 105.i | 117.u | |
| 10. | 22.□ | 34." | 46.. | 58.: | 70.F | 82.R | 94.^ | 106.j | 118.v | |
| 11. | 23.□ | 35.# | 47./ | 59.; | 71.G | 83.S | 95._ | 107.k | 119.w | |
| 12. | 24.□ | 36.$ | 48.0 | 60.< | 72.H | 84.T | 96.` | 108.l | 120.x | |

**Figure 6.5 ASCII table's first 128 characters**

In figure 6.5, ISO/IEC 10646 character set's first 128 characters is showed. They are also same with ASCII table's first 128 characters

**Table 6.84 Using character entities**

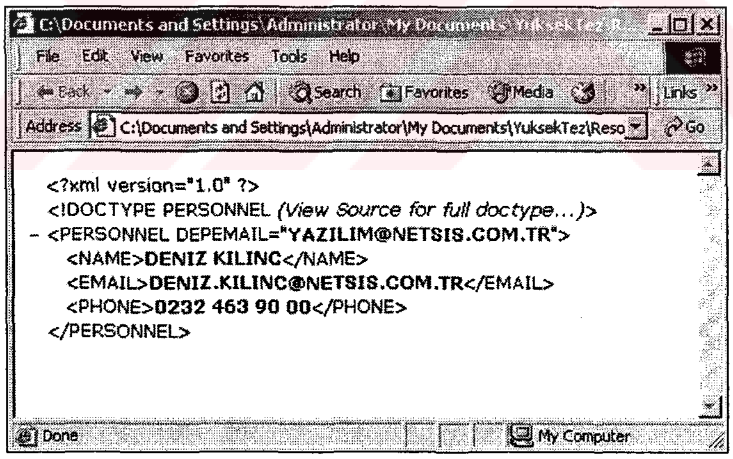
```
<?xml version="1.0"?>
<!DOCTYPE PERSONNEL[
  <!ELEMENT PERSONNEL (NAME,EMAIL,PHONE)>
  <!ELEMENT NAME (#PCDATA)> <!ELEMENT EMAIL (#PCDATA)>
  <!ELEMENT PHONE (#PCDATA)>
  <!ENTITY enName "&#75;&#73;&#76;&#73;&#78;&#67;">
  <!ENTITY izmSectCode "&#48;&#50;&#51;&#50;">
  <!ATTLIST PERSONNEL
              DEPEMAIL CDATA  "YAZILIM&#64;NETSIS.COM.TR">
]>
<PERSONNEL>
```

**Table 6.84 Continued…**

```
  <NAME>&#68;&#69;&#78;&#73;&#90; &enName;</NAME>
   <EMAIL>
      &#68;&#69;&#78;&#73;&#90;&#46;&enName;&#64;NETSIS.COM.TR
   </EMAIL>
  <PHONE>&izmSectCode; 463 90 00</PHONE>
</PERSONNEL>
```

In table 6.84, three types of character entity usage is described;

- *Using character entities in element declarations*, in NAME and EMAIL element definitions, characters entities are used. The whole characters of DENIZ name is constituted from 5 character entities.
- *Using character entities in attribute declarations*, PERSONNEL element's DEPEMAIL attribute includes the character entity which is mapped to "&".
- *Using character entities in other character entity definitions*, enName entity is constituted from character entities which has KILINC name.


C:\Documents and Settings\Administrator\My Documents\YuksekTez\R
File Edit View Favorites Tools Help
Back Search Favorites Media Links
Address C:\Documents and Settings\Administrator\My Documents\YuksekTez\Reso
Go
Done
My Computer

```
<?xml version="1.0" ?>
<!DOCTYPE PERSONNEL (View Source for full doctype...)>
- <PERSONNEL DEPEMAIL="YAZILIM@NETSIS.COM.TR">
    <NAME>DENIZ KILINC</NAME>
    <EMAIL>DENIZ.KILINC@NETSIS.COM.TR</EMAIL>
    <PHONE>0232 463 90 00</PHONE>
  </PERSONNEL>
```

**Figure 6.6 Web rendering of XML documents which have character entities**

In figure 6.6, web rendering of and XML documents is showed which have character entities.

## 6.16 Language Identifiers

In document processing, it is often useful to identify the natural or formal language in which the content is written. A special attribute named xml:lang may be inserted in documents to specify the language used in the contents and attribute values of any element in an XML document (WEB_12, 2004). In valid documents, this attribute, like any other, must be declared if it is used. The values of the attribute are language identifiers as defined by [IETF RFC 1766]. Language identifiers may have three types of value;

- A two-letter language code as defined by [ISO 639], "Codes for the representation of names of languages" (WEB_12, 2004).

| | |
|---|---|
| En (English) | Tr (Turkish) |
| Tr (Turkish) | JP (Japanese) |
| TR (Turkish) | tR (Turkish) |
| FR (French) | Fr (French) |

- A language identifier registered with the IANA (Internet Assigned Numbers Authority). Begins with the -i or -I prefixes (WEB_12, 2004).

| | |
|---|---|
| i-no-bok | i-navajo |
| i-no-nyn | i-mingo |

- A language identifier assigned by the user, or agreed on between parties in private use; these must begin with the prefix "x-" or "X-" in order to ensure that they do not conflict with names later standardized or registered with IANA (WEB_12, 2004).

| | |
|---|---|
| x-klingon | X-Elvish |

**Table 6.85 Language identifiers**

```
<xml version="1.0">
<list>
  <p xml:lang="en-GB ">What colour is it?</p>
  <p xml:lang="en-US ">What color is it?</p>
  <p xml:lang="en-GB ">Normalization</p>
</list>
```

## 6.17 Document Validation Constraints

When an XML document matches or W3C validation rules and constraints, it is named "valid". Some of these constraints are below;

- DTD name and root element name must be same.
- Parameter entity definitions' orders and locations in XML documents must be well-arranged.
- If there exist attribute default values which reference to external DTD documents, XML declaration's *standalone* attribute can not be *no*.
- If EMPTY keyword is used, element must not include content.
- If *ANY* keyword is used, all children's types, sequences must be defined properly.
- If Mixed keyword is used the content of element must include both character data and child elements. All children's types, sequences must be defined properly.
- Attribute value must match the attribute's type.
- An attribute can be defined only once.
- If element type declarations are defined using parameter entity references, reference text can not be empty.
- ID attribute's value can be able to identify its attached element from other elements.
- An element can have only one ID attribute.
- Elements' ID attributes must have the #IMPLIED or #REQUIRED attribute defaults.
- IDREF attribute's value must reference ID attribute's value. And IDREFS attribute must be constituted from more than one IDREF.

- An attribute value which is in the ENTITY type must match one of the defined notations and all notation names must be declared in DTD mechanism.
- If an attribute default is defined using #REQUIRED keyword, defined element must have this attribute in XML document.
- Default attribute value must match the attribute's type.
- If an attribute default is defined using #FIXED keyword, defined element's attribute can only have this default value.

# CHAPTER SEVEN
# INTRODUCTION TO XSL TECHNOLOGY

## 7.1 Introduction

XSL (Extensible Stylesheet Language) technology is a derivate of XML technology. XSL technology's development goal is to format and transform XML documents to another type of documents like HTML, WML (WEB_42, 2004), and PDF (WEB_16, 2004). The processing logic of an XSL parser is like below;

- An XML input document is passed to XSL document.
- XML document is processed using XSL technology.
- A result document is constituted.
- Although XSL looks like CSS technology, it supports many features that CSS cannot success.

## 7.2 XSL (Extensible Stylesheet Language)

XSL (Extensible Stylesheet Language) technology (Gardner C. & Rendon Z., 2002) is created using XML language. With another definition, it is an XML application. XSL technology seems CSS technology that formats HTML pages. However, XSL is more powerful and flexible than CSS. While CSS can only make changes about presentations (font, color, border), XSL technology's functionalities are below;

- *Filtering,* Provides to be chosen only worked data. For example, stocks with the percentage18 value added tax could be wanted to query.
- *Arithmetic Calculation,* Arithmetic summary, multiplication, division and subtraction processes can be done over XML data. For example; by adding stock entry and exit transactions, stock balance amounts can be calculated.
- *Composition,* More than one XML documents can be combined in one document. For example, stock transactions of different areas can be composed in one XML document.
- *Ordering,* XML elements can be ordered within some rules. For example, a stock report in which all elements are ordered according to their names.

XSL provides to make each type of query and transformation by accessing all XML units like elements, attributes, comments and processing instructions (WEB_16, 2004). XSLT and XPath are used technologies within XSL language. That is to say, XSLT and XPath are sub parts of XSL.

## 7.3 XSLT (Extensible Stylesheet Language Transformations)

XSLT language provides XML documents to be transformed other XML (WEB_16, 2004) documents, HTML pages, WML pages, PDF (Portable Document Format) (WEB_40, 2004) and CSV like formatted documents and programming language source files that are written in Delphi or Java. During transformation process, rules and declarations in the XSL file are used. Most popular transformation usages of XSL are HTML and WML produced ones. For example, WML pages provide to render data via WAP (Wireless Application Protocol) (WEB_41, 2004) browsers. WML language is developed using XML technology moreover, with its structure it is possible to transform XML documents to WML pages (WEB_42, 2004) using XSLT technology.

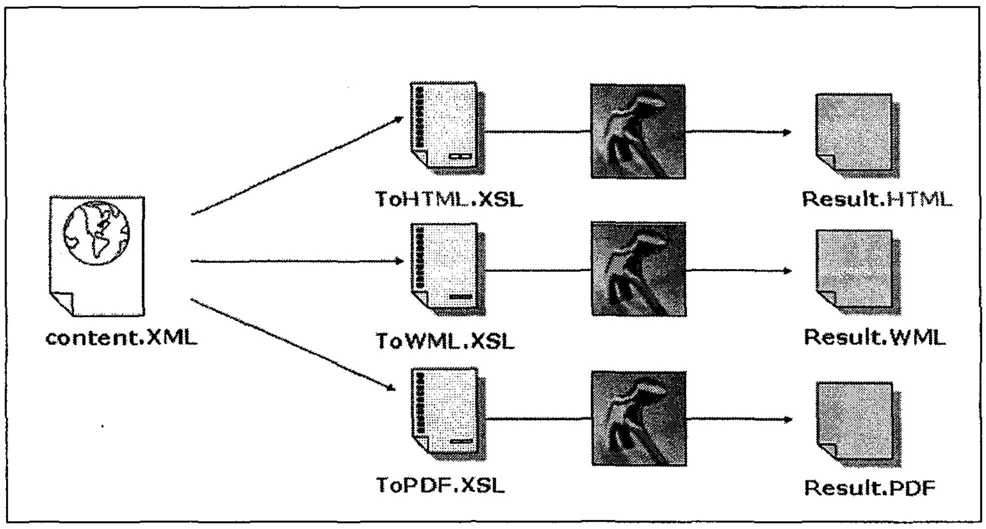


**Figure 7.1 XSLT Transformations**

## 7.4 First XSL Example

It is always useful to start with, a simple and easy sample to understand how XML documents are converted and transformed to other formats using XSL and XSLT.

**Table 7.1 First XSL and XSLT transformation**

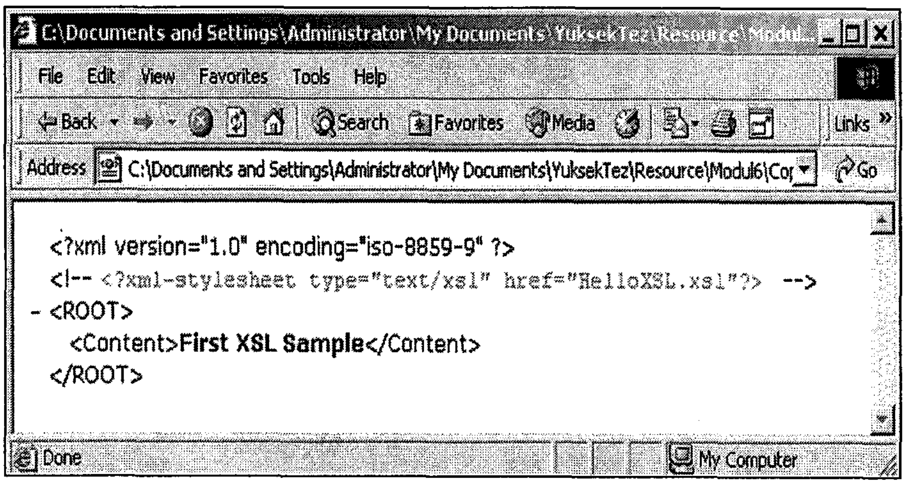
| **[HelloXSL.xml]** |
|---|
| <?xml version="1.0" encoding="iso-8859-9"?><br>*<?xml-stylesheet type="text/xsl" href="HelloXSL.xsl"?>*<br><ROOT><br>  <Content>First XSL Sample</Content><br></ROOT> |
| **[HelloXSL.xsl]** |
| <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"<br>                version="1.0"><br> <xsl:output method="html"/><br> <xsl:template match="/"><br>   <HTML><br>     <BODY><br>       <table border="1"><br>       <tr><td bgcolor="silver">/ROOT/Content Query Result</td><br>         <td bgcolor="silver"><br>         <font color="red"><B>*<xsl:value-of select="/ROOT/Content"/>*</B><br>         </font> |

**Table 7.1 Continued…**

```
      </td></tr>
     </table>
    </BODY>
   </HTML>
 </xsl:template>
</xsl:stylesheet>
```

In table 7.1, *HelloXSL.xml* content document and a transformation file with the name of *HelloXSL.xsl* are showed. XML document is transformed to an HTML file using XSLT technology. As it seen, XSL file includes many HTML elements and all of them must be well-formed unlike in HTML pages. For example; each element must have start and tags which are case sensitive. <xsl:stylesheet> is the root element of an XSL file that contain XSL templates and XSLT transformation rules (WEB_16, 2004). The root element's *xmlns:xsl* attribute is the namespace that identifies which suitable XSL elements can be used within document. x*sl* key after *xmlns* literal specifies all XSL elements must start with *xsl:* prefix in the XSL file.

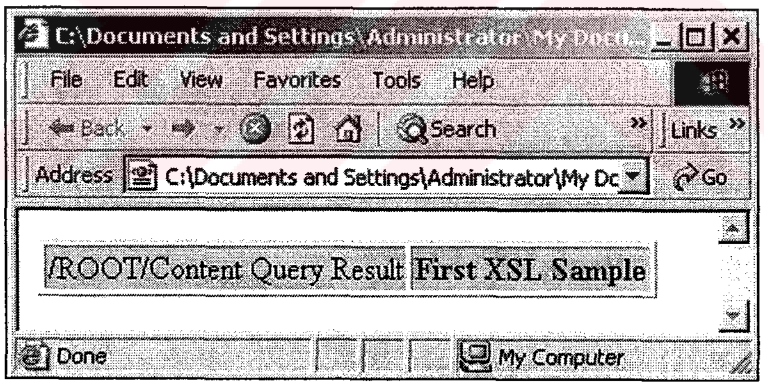
XSL files may include one or more templates in the <xsl:stylesheet> root element. Templates are defined using <xsl:template> elements. Templates' *match* attribute contains an XPath specification query to access XML units by starting from the root element. *match="/"* attribute value provides to be chosen all elements including the XML root element (Gardner C. & Rendon Z., 2002). With other words, it provides matching of document and XSL template one by one.


C:\Documents and Settings\Administrator\My Documents\YuksekTez\Resource\Modul...
File Edit View Favorites Tools Help
Back Search Favorites Media Links
Address C:\Documents and Settings\Administrator\My Documents\YuksekTez\Resource\Modul6\Co
Go
Done
My Computer

```
  <?xml version="1.0" encoding="iso-8859-9" ?>
  <!-- <?xml-stylesheet type="text/xsl" href="HelloXSL.xsl"?> -->
- <ROOT>
    <Content>First XSL Sample</Content>
  </ROOT>
```

**Figure 7.2 First XSL without reference**

In figure 7.2, first XSL sample is tried to render using Internet Explorer web browser, but style-sheet reference line of XML document is closed using comment line characters. So rendering will not be successful.


C:\Documents and Settings\Administrator\My Docu...
File Edit View Favorites Tools Help
Back Search Links
Address C:\Documents and Settings\Administrator\My Dc
Go
Done
My Computer

| /ROOT/Content Query Result | First XSL Sample |
|---|---|

**Figure 7.3 Simple transformation output**

In figure 7.3, *HelloXSL.xml* document is rendered and transformed to HTML form in the Internet Explorer web browser. This presentation with XSL is more readable than before.

## 7.5 XSLT Transformation and XML Document Structures

To achieve transformation process, XSLT creates a tree structure by parsing the XML document (Gardner C. & Rendon Z., 2002). Then it traverses tree starting from the root element. Finally, transformation process is done according to rules in the XSL file.

While XSLT processor traverses tree structure, it uses XPath declarations to access tree's different locations and nodes. That is to say, XPath technology is used to address the nodes in the created XML tree and provides to walk around chosen tree nodes (Gardner C. & Rendon Z., 2002).

The nodes in the tree structure are elements, attributes, element-attribute text, comment lines and processing instruction lines. At the top of tree, root element is located and sub-nodes are placed at bottom (WEB_16, 2004). XPath technology describes special nodes in tree to which parser can access. These node types are below;

- *Root node* symbolizes the root element that contains whole XML document. There can not be other nodes above the root element (Gardner C. & Rendon Z., 2002).
- *Element node* symbolizes all elements include root node. Element nodes may have sub or upper nodes. Although the root node is an element node at the same time, it has no upper elements. Lower node can also include other nodes or text nodes (Gardner C. & Rendon Z., 2002).
- *Attribute node* symbolizes all elements' attributes in an XML document. Attribute nodes are located at the lower level than the elements to which they belong to or at the same level with the elements' text nodes. Unless it is not accessed an attribute directly from schema or DTD, no attribute node is added to XML document (Gardner C. & Rendon Z., 2002).
- *Text node* symbolizes all elements' contents in an XML document. If element has CDATA sections, their content are marked as text nodes. If element has both CDATA sections and text content, CDATA sections'

contents are added to text content. When text content has one or more character or entity references, firstly entities are replaced their real values and then text node is constituted (Gardner C. & Rendon Z., 2002).

- *Comment node* symbolizes comment lines in the document. These comment lines must be located in the XML document but out of the DTD or schema (Gardner C. & Rendon Z., 2002).
- *PI node* symbolizes processing instruction lines in the document. This node includes PI's name and parameters (Gardner C. & Rendon Z., 2002).

## 7.6 XSL Stylesheet Structures

XSL stylesheet pages that provide to transform XML pages to other page formats are constituted from three sections;

- XSL declaration
- Declaration of transformation type
- Templates and XPath rules

### 7.6.1 XSL Declaration

XSL pages can start with XML declaration line. However, it is not an obligatory to use XML declarations. XSL declaration is an attribute in <xsl:stylesheet> root element (WEB_16, 2004) and has an XSL namespace in addition to version information which is also included in XML declarations. XSL pages are saved with the *".xsl"* extension as standard. Web browsers know this type of extension and format their content.

W3C organization has two types of XSLT version. Each version's name-space is different. Versions' name-spaces are in the form of URI (Uniform Resource Identifier) standard. *xsl* key after *xmlns* (WEB_16,2004) literal specifies all XSL elements must start with *xsl:* prefix in the XSL file.

**Table 7.2 XSL declaration**

| **XSLT 1.0 version** |
|---|
| <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"<br>version="1.0"><br>...<br></xsl:stylesheet> |
| **XSLT 1.1 version** |
| <xsl:stylesheet xmlns:xsl="http://www.w3.org/2001/XSL/Transform"<br>version="1.1"><br>...<br></xsl:stylesheet> |

### 7.6.2 Declaration of Transformation Type

*output* element is used to specify XSLT transformation type. *method* attribute of output element may have some specific values that are also format names (WEB_16, 2004). Three format types are described;

- method=”**html**”, the transformation type is HTML.
- method=”**text**”, the transformation type is text characters or whatever programming languages source code.
- method=”**xml**”, the transformation type is SVG (Scalable Vector Graphics), MATHML, CML or WML like and XML formatted documents.

**Table 7.3 Transformation formats and attributes (WEB_16, 2004)**

| Attribute | Transformation Format | Description |
|---|---|---|
| Encoding | method=”html”,method=”xml”, method=”text” | Specifies language coding for character or text. |
| Doctype-system | method=”html”,method=”xml” | Specifies document descriptor. |
| doctype-public | method=”html”,method=”xml” | Specifies document general descriptor. |
| Version | method=”html”,method=”xml” | Specifies the version of XML or HTML document declarations. |
| Standalone | method=”xml” | Specifies whether or not an XML document can |

| | | |
|---|---|---|
| | | reference external resources. |
| media-type | method="html",method="xml", method="text" | Specifies the MIME (Multipurpose Internet Mail Extension) type of transformed document. |
| Method | | Specifies the format of transformed document. Html, XML or text can be used. |
| omit-xml-declaration | method="xml" | Specifies whether or not XML declaration will be discarded in the XML document which will be transformed. |
| Indent | method="html",method="xml" | Specifies the indentation of tags in the XML document, which will be transformed. |
| cdata-section-elements | method="xml" | Specifies the elements which must be included in CDATA sections. |

### 7.6.3 Determination of Transformation Format as HTML

To determine transformation format as HTML, "html" value is assigned to *output* element's *method* attribute. With the usage of this format, it is provided to choose HTML tags like <HTML>, <BODY> or <P> (WEB_16, 2004).

**Table 7.4 Usage of HTML transformation format**

**[XMLContent.xml]**

```
<?xml version="1.0"?>
<CAMERA_CATEGORIES>
  <CAMERA>
    <BRAND>Canon</BRAND>
    <MODEL>Z155</MODEL>
    <PRICE>1875000</PRICE>
    <WEIGHT>120 gr.</WEIGHT>
  </CAMERA>
</CAMERA_CATEGORIES>
```

**[XMLContent.xsl]**

**Table 7.4 Continued...**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
 <xsl:output method="html"/>
 <xsl:template match="/|*">
   <HTML>
     <BODY>
       <P>
         <xsl:value-of select="CAMERA_CATEGORIES/CAMERA/BRAND"/>
       </P>
     </BODY>
   </HTML>
 </xsl:template>
</xsl:stylesheet>
```

**[XMLContent.html]**

```
<HTML>
  <BODY>
    <P>Canon</P>
   </BODY>
</HTML>
```

In table 7.4, <HTML>, <BODY>, <P> HTML tags and XSLT template rules are used together. The main template is described using <xsl:template match="/|*"> XSL and XPath expressions. It is provided to access all element nodes and root node by using "/|*" XPath expression.

*<xsl:value-of select=*"CAMERA_CATEGORIES/CAMERA/BRAND"*/>* XPath expression provides to choose all brand elements of cameras.

**Table 7.5 XMLContent2.xsl document**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
 <xsl:output method="html"/>
 <xsl:template match="/|*">
   <HTML>
     <HEAD>
       <TITLE>XMLContent2.xsl page</TITLE>
       <STYLE TYPE="text/css">
```

**Table 7.5 Continued...**

```
        <xsl:comment>
          .trHead {background-color:silver; font-family:Verdana;
                      font-size:11;font-weight:bold}
          .trRow  {background-color:silver; font-family:Verdana;
                       font-size:11}
        </xsl:comment>
      </STYLE>
    </HEAD>
    <BODY>
      <TABLE>
        <TR><TD class="trHead">Brand</TD>
          <TD class="trRow">
           <xsl:value-of
            select="CAMERA_CATEGORIES/CAMERA/BRAND"/>
          </TD>
        </TR>
        <TR>
          <TD class="trHead">Model</TD>
           <TD class="trRow">
            <xsl:value-of
            select="CAMERA_CATEGORIES/CAMERA/MODEL"/>
           </TD>
        </TR>
        <TR><TD class="trHead">Price</TD>
          <TD class="trRow">
           <xsl:value-of
             select="CAMERA_CATEGORIES/CAMERA/MODEL"/>
          </TD>
        </TR>
      </TABLE>
    </BODY>
  </HTML>
 </xsl:template>
</xsl:stylesheet>
```

In table 7.5, *XMLContent.xml* document is transformed to an HTML document using XSLT transformation rules and XPath matching expressions all of which are defined in *XMLContent2.xsl* document. The most frequently used HTML tags <TABLE>, <TR> and <TD> are mixed with cascade style sheets and “trHead”, “trRow” classes are assigned to <TD> table cells. *<xsl:comment>* element is provided to be added string literals *"<!--"* and *"-->"*.

### 7.6.4 Determination of Transformation Format as Text

To determine transformation format as HTML, "text" value is assigned to *output* element's *method* attribute. Text is located between *"<xsl:text>"* and *"</xsl:text>"* elements. They are used together with other xsl elements, template rules and XPath expressions (WEB_16, 2004).

**Table 7.6 Determination of Transformation Format as Text**

**[TextContent.xml]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<?xml-stylesheet type="text/xsl" href="TextContent.xsl"?>
<BOOK>
  <HEADER>XML Technology</HEADER>
  <AUTHOR>Deniz KILINÇ</AUTHOR>
</BOOK>
```

**[TextContent.xsl]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
 <xsl:output method="text"/>
 <xsl:template match="/|*">
   <xsl:text>
     Writer of the Book:
   </xsl:text>
   <xsl:value-of select="BOOK/AUTHOR"/>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
Writer of the Book: Deniz KILINÇ
```

In table 7.6 *TextContent.xml* document is transformed to text output using *"TextContent.xsl"* XSL document. *<xsl:output method="text"/>* provides to select output format as native text.

XPath expressions (xsl:value-of select) in the main XSL template can not be included in the *"<xsl:text>"* element content which are added to document as native text. *"iso-8859-9"* encoding attribute value is chosen to preview Turkish characters in XML declaration.

It is possible to transform XML document to programming languages' source codes by using XSL rules. The main point in this type of transformation is to use programming language elements correctly with other text elements (WEB_16, 2004).

**Table 7.7 Determination of Transformation Format as Text - 2**

**[TextContent1.xml]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<?xml-stylesheet type="text/xsl" href="TextContent1.xsl"?>
<FUNCTION>
  <NAME>Carp</NAME>
  <RESULTTYPE>Double</RESULTTYPE>
  <PARAMTYPE1>Integer</PARAMTYPE1>
  <PARAMTYPE2>Integer</PARAMTYPE2>
</FUNCTION>
```

**[TextContent1.xsl]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              version="1.0">
 <xsl:output method="text"/>
 <xsl:template match="/|*">
  <xsl:text>Function </xsl:text><xsl:value-of select="FUNCTION/NAME"/>
  <xsl:text>(X:</xsl:text>
  <xsl:value-of select="FUNCTION/PARAMTYPE1"/><xsl:text>;Y:</xsl:text>
  <xsl:value-of select="FUNCTION/PARAMTYPE2"/><xsl:text>):</xsl:text>
  <xsl:value-of select="FUNCTION/RESULTTYPE"/><xsl:text>;</xsl:text>
  <xsl:text>
   Begin
    Result := X*Y;
   End;
  </xsl:text>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
Function Multiply(X:Integer; Y:Integer):Double;
```

**Table 7.7 Continued...**

```
Begin
 Result := X*Y;
End;
```

In table 7.7, a simple code section that is written in Pascal programming language is produced using XML and XSL technologies. Function gets two parameters in integer type and multiplication of them that is in double type is returned as output of the function.

## 7.6.5 XSL Templates

XSL uses XML to describe these rules, templates, and patterns. Template rules defined by the *xsl:template* (WEB_16, 2004) element is the most important part of the XSL style sheet. Each template rule is an xsl:template element which starts with *<xsl:template>* starting element and ends with *</xsl:template>* closing element (WEB_16, 2004). These associate particular output with particular input. Each xsl:template element has a match attribute that specifies which nodes of the input document the template is instantiated for.

Each of XSL template rules are also an XSL element. In addition to XSL rules, templates may have native text. However, both native text and template rules must be exactly well-formed.

**Table 7.8 Simple template usage**

```
<?xml version="1.0" encoding="iso-8859-9"?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                 version="1.0">
  <xsl:output method="html">
  <xsl:template match="/">
   <HTML>
     <BODY>
      Simple template usage which has no XSL rule
     </BODY>
```

**Table 7.8 Continued...**

```
  </HTML>
 </xsl:template>
</xsl:stylesheet>
```

In table 7.8, one template rule is showed. This template matches XML document's root element by assigning the "/" value to match attribute. Although XSL template has no additional pattern rule or xsl element (value-of select etc.), it contains some HTML specific tags and native text data.

The *xsl:apply-templates* rule (Gardner C. & Rendon Z., 2002) inserts the text of the matched source element into the output document. In general, to include content in the child nodes, you have to recursively process the nodes through the XML document. The element that does this is xsl:apply-templates . By including xsl:apply-templates in the output template, you tell the formatter to compare each child element of the matched source element against the templates in the style sheet; and, if a match is found, output the template for the matched node.

**Table 7.9 Traversing elements iteratively with xsl:apply-templates element**

**[UseTemplate.xml]**

```
<?xml-stylesheet type="text/xsl" href="UseTemplate.xsl"?>
<?xml version="1.0"?>
<CAMERA_CATEGORIES>
 <CAMERA>
  <BRAND>Canon</BRAND>
  <MODEL>Z155</MODEL>
 </CAMERA>
 <CAMERA>
  <BRAND>Nikon</BRAND>
  <MODEL>S5Z</MODEL>
 </CAMERA>
</CAMERA_CATEGORIES>
```

**[UseTemplate.xsl]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
 <!--1.Iteration-->
```

**Table 7.9 Continued...**

```
 <xsl:template match="/">
   <HTML>
     <BODY>
          <xsl:apply-templates/>
      </BODY>
   </HTML>
 </xsl:template>
 <!--2.Iteration-->
 <xsl:template match="CAMERA_CATEGORIES">
   <TABLE BORDER="1">
          <xsl:apply-templates/>
   </TABLE>
 </xsl:template>
 <!--3.Iteration-->
 <xsl:template match="CAMERA">
   <TR>
       <TD>Each camera</TD>
       <TD>
          <xsl:apply-templates select="BRAND"/>-
          <xsl:apply-templates select="MODEL"/>
       </TD>
   </TR>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
<HTML>
   <BODY>
      <TABLE BORDER="1">
         <TR><TD>Each camera</TD><TD>Canon-Z155</TD></TR>
         <TR><TD>Each camera</TD><TD>Canon-Z155</TD></TR>
      </TABLE>
   </BODY>
</HTML>
```

In table 7.9, XSL document has three XSL templates. Each template works with connected to each other to provide iteration. XSL document's processing steps by XSLT parser is below;

- In the first template XML document's root element is matched by assigning the "/" value to match attribute. For now, there is no iteration process.
- *<HTML>* and *<BODY>* opening tags are added to output document.

- First *<xsl:apply-templates/>* element is the start point of iteration. Root element (CAMERA_CATEGORIES) is begin to be processed
  - A template is searched that is within root element. Since second template is in the root template. Second template is begin to be processed and *<TABLE BORDER="1">* HTML element is added to output document.
    - Second *<xsl:apply-templates/>* element specifies the iteration process over the child elements' of root element. When it is meeting with the first child element named CAMERA, a new template is searched within the XSL document that belongs to CAMERA.
    - Third *<xsl:apply-templates/>* element that belongs to the CAMERA element is found and HTML elements *<TR>* *<TD>Each camera</TD>* is added to output. Then first CAMERA element's *BRAND* and *MODEL* child elements' content and *</TD></TR>* HTML elements are added to output.
    - The same process is executed for the second CAMERA element.
  - *</TABLE>* HTML element is added to output document.
- Finally *</BODY>* and *</HTML>* closing HTML elements are added to output document.

## 7.6.6 <xsl:value-of select> Element

Value-of select (WEB_16, 2004) element copies the element node's value that is in the input document into the output document. Select attribute's value is an XPath expression and specifies the taken value.

**Table 7.10 xsl:value-of element**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
  <xsl:template match="CAMERA_CATEGORIES">
    <xsl:apply-templates/>
  </xsl:template>
```

**Table 7.10 Continued...**

```
  <xsl:template match="CAMERA">
    <xsl:value-of select="BRAND"/>
    <xsl:value-of select="MODEL"/>
  </xsl:template>
 </xsl:stylesheet>
```

In table 7.10, two XSL templates are used. In the second template, BRAND and MODEL elements that belong to the CAMERA element are chosen and their values are added to output document by using *value-of select* XSL element.

## 7.6.7 Processing XML Elements Iteratively

Iterative processing is the one of the most important processing technique in all programming languages. In XSL technology iterative or recursive processes can be carried out in two ways (Harold E.R., 1999);

- By using XSL template technology and *<xsl:apply-templates>* element that is showed in table 7.9.
- By using <xsl:for-each> element.

For-each statement is a standard looping command common to most programming languages, which instructs the parser to loop through all the element children matching against the value of the *select* attribute (Harold E.R., 1999).

**Table 7.11 xsl:for-each element**

**[UseIteration.xml]**

```
<?xml-stylesheet type="text/xsl" href="UseIteration.xsl"?>
<?xml version="1.0"?>
<CAMERAS>
 <CAMERA>
  <BRAND>Canon</BRAND>
  <MODEL>Z155</MODEL>
  <PRICE>1000000</PRICE>
  <WEIGHT>120 gr.</WEIGHT>
```

**Table 7.11 Continued…**

```
 </CAMERA>
 <CAMERA>
   <BRAND>Nikon</BRAND>
   <MODEL>S5Z</MODEL>
   <PRICE>1500000</PRICE>
   <WEIGHT>130 gr.</WEIGHT>
 </CAMERA>
 </CAMERAS>
```

**[UseIteration.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           version="1.0">
 <xsl:template match="/CAMERAS">
   <HTML>
    <HEAD><TITLE>Selling Cameras</TITLE></HEAD>
    <BODY>
    <TABLE>
     <xsl:for-each select="CAMERA">
       <TR>
         <TD><xsl:value-of select="BRAND"/></TD>
         <TD><xsl:value-of select="MODEL"/></TD>
         <TD><xsl:value-of select="PRICE"/></TD>
         <TD><xsl:value-of select="WEIGHT"/></TD>
       </TR>
     </xsl:for-each>
    </TABLE>
    </BODY>
   </HTML>
 </xsl:template>
 </xsl:stylesheet>
```

**[Output]**

```
 <HTML>
    <HEAD><TITLE>Selling Cameras</TITLE></HEAD>
    <BODY>
    <TABLE>
       <TR> <TD>Canon</TD><TD>Z155</TD>
             <TD>1000000</TD><TD>120 gr.</TD>
       </TR>
       <TR><TD>Nikon</TD><TD>S5Z</TD>
             <TD>1500000</TD><TD>130 gr.</TD>
       </TR>
    </TABLE>
    </BODY>
 </HTML>
```

In table 7.11, XML document template pattern is constituted by choosing the CAMERAS element. By using *<xsl:for-each select="CAMERA">* XSL rule, all <CAMERA> elements are processes recursively in a loop that can be considered as a node list.

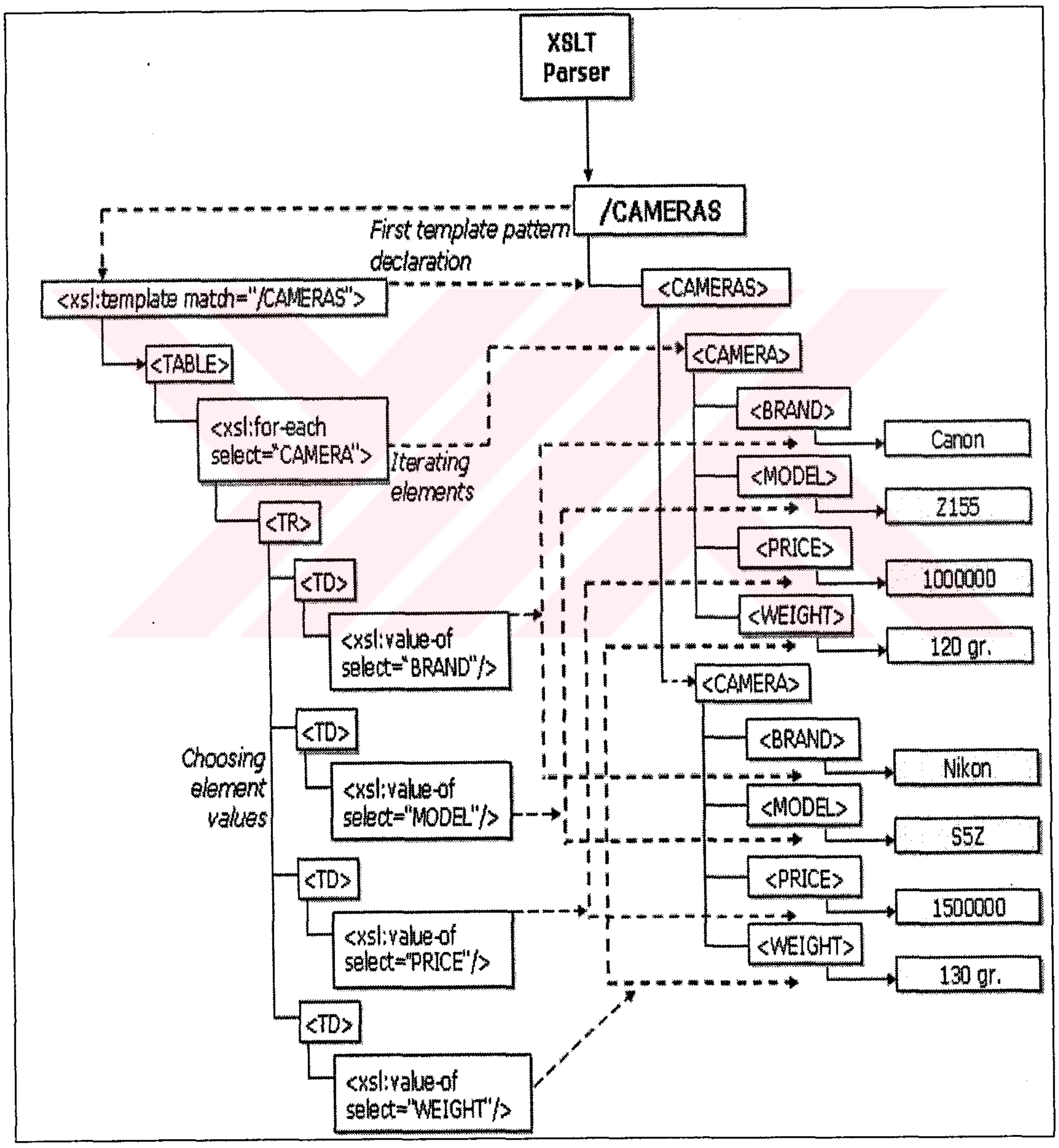

XSLT Parser
/CAMERAS
First template pattern declaration
<xsl:template match="/CAMERAS">
<CAMERAS>
<TABLE>
<CAMERA>
<xsl:for-each select="CAMERA">
Iterating elements
<BRAND>
Canon
<MODEL>
Z155
<TR>
<PRICE>
1000000
<TD>
<WEIGHT>
<xsl:value-of select="BRAND"/>
120 gr.
<CAMERA>
<TD>
<BRAND>
Choosing element values
Nikon
<xsl:value-of select="MODEL"/>
<MODEL>
S5Z
<TD>
<PRICE>
1500000
<xsl:value-of select="PRICE"/>
<WEIGHT>
130 gr.
<TD>
<xsl:value-of select="WEIGHT"/>

**Figure 7.4 XSLT processing steps**

In figure 7.4, XSLT processing mechanism of the example in table 7.11 is showed by XSLT parser. XSLT parser behaves different for each XSL element or rule. XSLT processing steps are below;

- CAMERAS root node is chosen with the first template pattern matching.
- <TABLE> HTML element is added to output document.
- CAMERA sub elements is started to be processed iteratively.
  - CAMERA element's BRAND, MODEL, PRICE and WEIGHT child nodes' values are added to output document with <TR> and <TD> HTML elements..
- </TABLE> closing HTML element is added to output document.
- Processing is finished.

### 7.6.8 Conditional Processing

Conditional processing provides to query XML elements by using specific conditions. They look like *if-then*, *if-then-else* and *switch-case* structures that are used in traditional programming languages. XSL provides two elements that allow you to change the output based on the input;

- By using <xsl:if> element (WEB_16, 2004).
- By using <xsl:choose>, <xsl:when>, <xsl:otherwise> elements (WEB_16, 2004).

Most of what you can do with xsl:if and xsl:choose can also be done by suitable application of templates. However, sometimes the solution with xsl:if or xsl:choose is simpler and more obvious.

### 7.6.9 <xsl:if> Element Usage

The <xsl:if> element provides if-then functionality, similar to if-then clauses found in programming languages like Delphi, Microsoft Visual Basic. <xsl:if> conditional processing element provides a simple facility for changing the output based on a pattern. The test attribute of xsl:if contains a select expression that evaluates to a Boolean. If the expression is true, the contents of the xsl:if element are output (Harold E.R., 1999). Otherwise, they are not.

**Table 7.12 xsl:if conditional processing element**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          version="1.0">
 <xsl:template match="/">
  <TABLE>
   <TR><TH>STATE</TH><TH>BRAND</TH><TH>MODEL</TH>
         <TH>PRICE</TH>  <TH>WEIGHT</TH>
   </TR>
   <xsl:for-each select="CAMERAS/CAMERA">
    <xsl:if test="PRICE &gt; 1000000">
     <TR bgcolor="red">
      <TD>Expensive</TD>
      <TD><xsl:value-of select="BRAND"/></TD>
      <TD><xsl:value-of select="MODEL"/></TD>
      <TD><xsl:value-of select="PRICE"/></TD>
      <TD><xsl:value-of select="WEIGHT"/></TD>
     </TR>
    </xsl:if>
   </xsl:for-each>
  </TABLE>
 </xsl:template>
</xsl:stylesheet>
```

In table 7.12, <xsl:if> conditional processing element's usage is showed. This condition provides to be chosen CAMERAS whose prices are greater than 1000000. To constitute condition, *PRICE &gt; 1000000* value is assigned to *test* attribute. &gt; predefined entity reference is used instead of mathematical operator > (greater than). The other samples of <xsl:if> elements are showed below;

- *<xsl:if test="CAMERAS/CAMERA">*;

  If one or more CAMERAS element and at the lower level of it one or more CAMERA element is existed, then conditional define is true.
- *<xsl:if test="count(CAMERAS/CAMERA) &gt;= 3">*

  If the number of CAMERA element under the CAMERAS root element is equal or greater than 3, then conditional define is true.
- *<xsl:if test=".//ADI">*;

  If the chosen element contains one or more ADI sub element, then conditional define is true.
- <xsl:if test="number(CAMERA/PRICE) ">

  If the PRICE element's value is numeric that is under the CAMERA element, then conditional define is true. If element value contains characters like *aeiou*, then conditional define is false.
- <xsl:if test="string(CAMERA/MODEL) ">

  If the MODEL element's value is string that is under the CAMERA element, then conditional define is true.

### 7.6.10 <xsl:choose> Element Usage

If <xsl:if> element is used, you can not specify the condition if the condition is not true. There are no <xsl:else> or <xsl:else-if> elements. The <xsl:choose>, <xsl:when> and <xsl:otherwise> elements (WEB_16, 2004) provide this functionality. The <xsl:choose> element selects one of several possible outputs depending on several possible conditions.

Each condition and its associated output template is provided by an <xsl:when> child element. <xsl:when> element is technically same with <xsl:if> element. The test attribute of the <xsl:when> element is a select expression with a Boolean value. If multiple conditions are true, only the first true one is instantiated. If none of the

<xsl:when> elements are true, the contents of the <xsl:otherwise> child elements are instantiated (Harold E.R., 1999).

**Table 7.13 xsl:choose conditional processing element skeleton**

```
<xsl:choose>
   <xsl:when test="Conditional Define-1">
   ...
   </xsl:when>
   <xsl:when test="Conditional Define-2">
   ...
   </xsl:when>
   <xsl:otherwise>
   ...
   </ xsl:otherwise>
</xsl:choose>
```

In table 7.13, the main skeleton of <xsl:choose> element is showed. At the most outside <xsl:choose> element is located. Under that <xsl:when> and <xsl:otherwise> are used.

**Table 7.14 xsl:choose element**

**[xsl-choose.xml]**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<?xml-stylesheet type="text/xsl" href="xsl-choose.xsl"?>
<STOCK_MODULE>
 <STOCK_TRANSACTIONS>
  <TRANSACTION NUM="1" OPR_TYPE="U" TYPE="INPUT">
   <DOCNO>00001</DOCNO>
   <DESCRIPTION>PRODUCTION S001 STOCK</DESCRIPTION>
   <QUANTITY>5</QUANTITY>
  </TRANSACTION>
  <TRANSACTION NUM="2" OPR_TYPE="F" TYPE="OUTPUT">
   <DOCNO>00002</DOCNO>
   <DESCRIPTION>SELLING S002 STOCK</DESCRIPTION>
   <QUANTITY>2</QUANTITY>
  </TRANSACTION>
  <TRANSACTION NUM="3" OPR_TYPE="E" TYPE="OUTPUT">
   <DOCNO>00003</DOCNO>
   <DESCRIPTION>MANUAL STOCK TRANSACTION</DESCRIPTION>
   <QUANTITY>5</QUANTITY>
```

**Table 7.14 Continued**

```
  </TRANSACTION>
 </STOCK_TRANSACTIONS>
</STOCK_MODULE>
```

**[xsl-choose.xsl]**

```
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           version="1.0">
 <xsl:template match="STOCK_MODULE">
  <TABLE>
   <xsl:for-each select="STOCK_TRANSACTIONS/TRANSACTION">
    <xsl:choose>
     <xsl:when test="@OPR_TYPE='U'">
      <TR bgcolor="lightblue">
       <TD>Production:
           <xsl:value-of select="DOCNO"/>:
           <xsl:value-of select="DESCRIPTION"/></TD>
      </TR>
     </xsl:when>
     <xsl:when test="@OPR_TYPE='F'">
      <TR bgcolor="silver">
       <TD>Selling:
            <xsl:value-of select="DOCNO"/>:
            <xsl:value-of select="DESCRIPTION"/></TD>
      </TR>
     </xsl:when>
     <xsl:otherwise>
      <TR bgcolor="lightbrown">
       <TD>Manual entry:
             <xsl:value-of select="DOCNO"/>:
             <xsl:value-of select="DESCRIPTION"/></TD>
      </TR>
     </xsl:otherwise>
    </xsl:choose>
   </xsl:for-each>
  </TABLE>
 </xsl:template>
 </xsl:stylesheet>
```

**[Output]**

```
<TABLE>
 <TR bgcolor="lightblue">
  <TD>Production: 00001: PRODUCTION S001 STOCK
   </TD>
 </TR>
 <TR bgcolor="silver">
```

**Table 7.14 Continued...**

```
   <TD>Selling: 00002: SELLING S002 STOCK
   </TD>
 </TR>
 <TR bgcolor="lightbrown">
  <TD>Manual entry: 00003: MANUAL STOCK TRANSACTION
   </TD>
 </TR>
</TABLE>
```

In table 7.14, *xsl-choose.xml* XML document that contains stock transaction information, *xsl-choose.xsl* XSL document that realizes matching process according to transaction type and output document of the XSLT transformation is showed. XSLT transformation is made by controlling the stock transaction's OPR_TYPE attribute.;

- OPR_TYPE='U' means production,
  - **<xsl:when** test="@OPR_TYPE='U'">
- OPR_TYPE='F' means selling,
  - **<xsl:when** test="@OPR_TYPE='F'">
- Transaction that has other attribute values are manual stock transactions.
  - **<xsl:otherwise>**

### 7.6.11 Adding PIs to Output Document - xsl:processing-instruction

To add processing instructions to the output document during the transformation process, <xsl:processing-instruction> XSL element (WEB_16, 2004) must be used. Generally, when output is an XML document, this XSL element is needed. The added processing instruction's attributes must also be attached to the declaration. But dynamic attribute assigning method (<xsl:attribute>) can not be used. The target of the processing instruction is specified by a required name attribute.

**Table 7.15 xsl:processing-instruction element**

**[PI.xml]**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<?xml-stylesheet type="text/xsl" href="PI.xsl"?>
<PARAMS>
  <PRM>I</PRM>
</PARAMS>
```

**[PI.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
 <xsl:output method="xml"/>
  <xsl:template match="/PARAMS">
    <xsl:processing-instruction name="params">
      param1="<xsl:value-of select="PRM"/>
    </xsl:processing-instruction>
    <TEMPREPORT> <CONTENT>...</CONTENT> </TEMPREPORT>
  </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
<?params param1="I"?>
<TEMPREPORT>
  <CONTENT>...</CONTENT>
</TEMPREPORT>
```

In table 7.15, a processing instruction with the name of *params* is added to XML output document by using <xsl:processing-instruction> XSL element. PI's attribute's value , named *param1*, is constituted by selection of the PRM sub-element from the XML input document.

### 7.6.12 Adding Comments to Output Document - xsl:comment

The <xsl:comment> element (WEB_16, 2004) inserts a comment in the output document. It has no attributes. Its contents are the text of the comment. It can not contain <xsl:attribute> or <xsl:element> XSLT elements which produce dynamic XSL elements and attributes.

**Table 7.16 xsl:comment element**

```
<xsl:comment>Usage of comment lines</xsl:comment>
[Output]
<!-- Usage of comment lines -->
```

### 7.6.13 Adding Elements to Output Document - xsl:element

There are two methods to add elements into the output document during XSLT transformation;

- ***Direct element adding;*** That is the most frequently used element adding method. E lements usually get inserted into the output document simply by using the literal elements themselves by matching the well-formedness constraints. For instance, to insert a table element you merely type <TABLE> and </TABLE>at the appropriate points in the style sheet (Harold E.R., 1999).
- ***<xsl:element> is used;*** This method is applied to insert elements into the output document. The name of the element is given by an attribute value template in the *name* attribute of <xsl:element>. It is especially used to transform XML documents to other XML types of XML documents (Harold E.R., 1999).

**Table 7.17 xsl:element usage**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
 <xsl:output method="xml"/>
 <xsl:template match="/STOCK_MODULE">
  <STOCK_OUTPUT_REPORT>
    <xsl:for-each select="STOCK_TRANSACTIONS/TRANSACTION">
      <xsl:if test="@TYPE='OUTPUT'">
        <xsl:element name="REPORT_TRANSACTION">
          <xsl:element name="TRANSACTION_DOC">
           <xsl:value-of select="DOCNO"/>
          </xsl:element>
          <TRANSACTION_QUANTITY>
```

**Table 7.17 Continued...**

```
        <xsl:value-of select="QUANTITY"/>
       </TRANSACTION_QUANTITY>
      </xsl:element>
    </xsl:if>
   </xsl:for-each>
  </STOCK_OUTPUT_REPORT>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
<STOCK_OUTPUT_REPORT>
 <REPORT_TRANSACTION>
   <TRANSACTION_DOC>00002</TRANSACTION_DOC>
   <TRANSACTION_QUANTITY>2</TRANSACTION_QUANTITY>
  </REPORT_TRANSACTION>
  <REPORT_TRANSACTION>
   <TRANSACTION_DOC>00003</TRANSACTION_DOC>
   <TRANSACTION_QUANTITY>5</TRANSACTION_QUANTITY>
  </REPORT_TRANSACTION>
</STOCK_OUTPUT_REPORT>
```

In table 7.17, *xsl-choose.xml* is filtered and transformed by using XSL document. Stock transactions that have the OUTPUT value in their TYPE attributes are chosen and transformed to another XML document. During transformation process two element adding methods are used.

<STOCK_OUTPUT_REPORT> and <TRANSACTION_QUANTITY> elements are added by using direct adding method. But, <REPORT_TRANSACTION> and <TRANSACTION_DOC> elements are added by using <xsl:element> XSL element.

### 7.6.14 Adding Attributes to Output Document - xsl:attribute

Attributes are name-value pairs (WEB_16, 2004). There are two methods to add attributes into the output document during XSLT transformation;

- ***Direct attribute adding;*** This method is not the most frequently used attribute adding method. Attributes usually get inserted into the output document simply by using the literal attributes themselves by matching the well-formedness constraints (Harold E.R., 1999).

- ***<xsl:attribute> is used;*** This method is applied to insert attributes into the output document by using <xsl:attribute> XSLT element. All xsl:attribute elements must come before any other content of their parent element. You cannot add an attribute to an element after you have already started writing out its contents (Harold E.R., 1999).

**Table 7.18 xsl:attribute usage**

**[attribute.xml]**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<?xml-stylesheet type="text/xsl" href="attribute.xsl"?>
<COLORS>
 <BACKGROUND>red</BACKGROUND>
</COLORS>
```

**[attribute.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           version="1.0">
   <xsl:template match="/">
    <BODY>
     <xsl:attribute name="bgcolor">
       <xsl:value-of select=" /COLORS/BACKGROUND" />
     </xsl:attribute>
    </BODY >
   </xsl:template>
</xsl:stylesheet>
```

In table 7.18, *bgcolor* attribute is added to BODY element by using the <xsl:attribute> element. *name* attribute's value of <xsl:attribute> element is the name of created attribute and it is chosen with the <xsl:value-of> XSL rule.

**Table 7.19 Direct attribute adding method**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           version="1.0">
   <xsl:template match="/">
     <BODY bgcolor="{/COLORS/BACKGROUND}">
     </BODY>
```

**Table 7.19 Continued...**

```
  </xsl:template>
</xsl:stylesheet>
```

In table 7.19, BODY element's bgcolor attribute value is an XPath expression. If carefully looked, XPath expression is not located as *<xsl:value-of select= "/COLORS/BACKGROUND"/>*. Instead of it *{/COLORS/BACKGROUND}* special expression is hold.

### 7.6.15 Adding Attribute Sets to Output Document - xsl:attribute-set

When it is needed to apply the same group of attributes to many different elements, of either the same or different classes, <xsl:attribute-set> (WEB_16, 2004) element must be used. <xsl:attribute-set> element is the main element of attribute-set and it can contain many attributes and each of which is defined using <xsl:attribute> element.

Technically, they are similar to CSS (Cascade Style Sheet) classes. *xsl:use-attribute-sets* element is used to assign an attribute-set to an element. If an element uses more than one attribute set, then all attributes from all the sets are applied to the element (Harold E.R., 1999). If more than one attribute set defines the same attribute with different values, then the one from the more important set is used.

**Table 7.20 xsl:attribute-set usage**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
 <xsl:attribute-set name="TABLE">
  <xsl:attribute name="bgcolor">silver</xsl:attribute>
  <xsl:attribute name="border">2</xsl:attribute>
 </xsl:attribute-set>
 <xsl:template match="/STOCK_MODULE">
  <table xsl:use-attribute-sets="TABLE">
   <xsl:for-each select="STOCK_TRANSACTIONS/TRANSACTION">
    <tr><td><xsl:value-of select="DOCNO"/></td></tr>
   </xsl:for-each>
```

**Table 7.20 Continued...**

```
 </table>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
<TABLE bgcolor="silver" border="2">...</TABLE>
```

In table 7.20, an attribute-set with the name of *TABLE* is created by using <xsl:attribute-set> XSLT element and assigned to HTML output's <TABLE> element. Attribute-set has *bgcolor* and *border* attributes.

### 7.6.16 Copying Document Nodes - xsl:copy

<xsl:copy> element copies nodes from source XML document into the output XML document during the XSLT transformation (WEB_16, 2004). Child elements, attributes, and other content are not automatically copied. However, the contents of the xsl:copy elements are an xsl:template element that can select these things to be copied as well.

This is often useful when transforming a document from one markup vocabulary to the same or a closely related markup vocabulary. One useful template xsl:copy makes possible is the identity transformation; that is, a transformation from a document into itself.

**Table 7.21 Identity transformation using xsl-copy**

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform
                version="1.0" ">
 <xsl:template match="*|@*|comment()|processing-instruction()|text()">
   <xsl:copy>
     <xsl:apply-templates select="*|@*|comment()|processing-
instruction()|text()"/>
   </xsl:copy>
 </xsl:template>
</xsl:stylesheet>
```

In table 7.21, an XSL document is showed that can make identity transformation. XPath expression "*|@*|comment()|processing-instruction()|text()" that is used in the transformation, copies source XML document including its all characters. For example, if comment lines and processing instructions are wished to exclude from destination document, it will be enough to exclude comment() and processing-instruction() XPath expressions.

### 7.6.17 Using Script Language – xsl:script, xsl:eval

During XSLT transformation, XSL rules and XPath expressions may not be adequate. In this type of circumstances, some programmatic approaches are needed instead of XSL rules. To solve this problem <xsl:script> and <xsl:eval> elements are used that supports both JavaScript and VBScript. Adding scripting capability to XSL enhances its effectiveness at the cost of portability (WEB_16, 2004).

**Table 7.22 Script language usage**

**[xsl-script.xml]**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<SALARYLIST>
  <PERSONNEL>
    <ID>0001</ID>
    <SALARY>3000000</SALARY>
  </PERSONNEL>
  <PERSONNEL>
    <ID>0002</ID>
    <SALARY>1000000</SALARY>
  </PERSONNEL>
</SALARYLIST>
```

**[xsl-script.xsl]**

```
<xsl:stylesheet
      language="VBScript" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/SALARYLIST">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>
```

**Table 7.22 Continued...**

```
 <xsl:template match="PERSONNEL">
   <xsl:copy>
     <xsl:apply-templates/>
   </xsl:copy>
 </xsl:template>
 <xsl:template match="ID">
   <xsl:copy>
     <xsl:apply-templates/>
     <xsl:eval>myValue(me)</xsl:eval>
   </xsl:copy>
 </xsl:template>
 <xsl:template match="SALARY">
   <xsl:copy>
     <xsl:apply-templates/>
     <xsl:eval>encrypt(me)</xsl:eval>
   </xsl:copy>
 </xsl:template>
 <xsl:script language="VBScript">
  Function encrypt(obj)
    encrypt = ((obj.nodeTypedValue * 5)/99999999*0.333333)/0.44
  End Function
  Function myValue(obj)
    myValue = (obj.nodeTypedValue)
  End Function
 </xsl:script>
</xsl:stylesheet>
```

**[Output]**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<SALARYLIST>
  <PERSONNEL>
    <ID>0001</ID>
    <SALARY>0.113636251136362</SALARY>
  </PERSONNEL>
  <PERSONNEL>
    <ID>0002</ID>
    <SALARY>3.78787503787875E-02</SALARY>
  </PERSONNEL>
</SALARYLIST>
```

In table 7.22, a simple XML document that keeps an entertainment's personnel's information and their salaries, an XSL document that encrypts personnel salaries and transforms them another XML document and the output XML document are showed.

The type of programming language is specified by assigning the *VBScript* value to the *language* attribute of *xsl:stylesheet* element.

To encrypt personnel salaries the *encrypt* function is written in VBScipt programming language. Function gets *me* object as input to process the content in its current location.

For example, the salary of the personnel which has the 0001 as its ID element's value is *<SALARY>3000000</SALARY>*. Actually, whole of this content is sent to the function as parameter. In the function is processed as an XML node in the type of *IXMLDOMNoade*. By using *nodeTypedValue* it is accessed to node value and after necessary multiplication and division processes, encrypted value is returned back.

Defined functions are called by using <xsl:eval> XSLT element. In the content of <xsl:eval> element, function's name and parameters must be written in a well-ordered manner (WEB_16, 2004).

### 7.6.18 Merging XSL Pages

You may need to use different standard style sheets for different vocabularies. However, you will also want style rules for particular documents as well. To reach external XSL resources, two basic XSL elements are used;

- <xsl:include>
- <xsl:import>

The xsl:import and xsl:include elements enable you to merge multiple style sheets so that you can organize and reuse style sheets for different vocabularies and purposes (WEB_16, 2004).

#### 7.6.18.1 Using <xsl:include> Element

The <xsl:include> element is a top level element that copies another style sheet into the current style sheet at the point where it occurs. (More precisely, it copies the contents of the xsl-stylesheet element in the remote document into the current document.) Its *href* attribute provides the URI or URN of the style sheet to include. An xsl:include element can occur anywhere at the top level after the last <xsl:import> element. (WEB_16, 2004)

**Table 7.23 Using <xsl:include> element**

**[sourceXSL.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
 <xsl:template match="/">
  <TABLE>
   <xsl:for-each select="CAMERAS/CAMERA">
    <xsl:if test="PRICE &gt; 1000000">
     <TR>
       <TD>Expensive</TD>
       <TD><xsl:value-of select="BRAND"/></TD>
       <TD><xsl:value-of select="MODEL"/></TD>
       <TD><xsl:value-of select="PRICE"/></TD>
       <TD><xsl:value-of select="WEIGHT"/></TD>
     </TR>
    </xsl:if>
   </xsl:for-each>
  </TABLE>
 </xsl:template>
 <xsl:include href="externalXSL.xsl"/>
</xsl:stylesheet>
```

**[externalXSL.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
 <xsl:template match="/">
  <TABLE>
    <TR><TD>External Resource</TD></TR>
  </TABLE>
 </xsl:template>
</xsl:stylesheet>
```

In table 7.23, <xsl:include> element is showed that provides to access to external XSL sources. The name of the XSL document to which it is wanted to reach, is *externalXSL.xsl* according to XSL source definition (href attribute value). And it must be located in the same place with XSL source document.

### 7.6.18.2 Using <xsl:import> Element

The <xsl:import> element is a top level element whose *href* attribute provides the URI of a style sheet to import (WEB_16, 2004). The difference of <xsl:import> element from <xsl:include> element that all <xsl:import> elements must appear before any other top level elements in the <xsl:stylesheet> root element.

**Table 7.24 Using <xsl:import> element**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
             version="1.0">
<xsl:import href="externalXSL.xsl"/>
<xsl:template match="/CAMERAS/CAMERA">
   <xsl:value-of select="BRAND"/>
  </TABLE>
 </xsl:template>
</xsl:stylesheet>
```

In table 7.24, <xsl:import> element is showed that provides to access to external XSL sources. <xsl:import> element is located under the root element. To reference external stylesheet, *externalXSL.xsl* value is assigned to *href* attribute's value.

### 7.6.19 <xsl:call-template> Element Usage

The <xsl:call-template> element is used to invoke a named template. This element appears in the contents of a template rule. It has a required *name* attribute that names the template it will call. When processed, the <xsl:call-template> element is replaced by the contents of the xsl:template element it names. (WEB_16, 2004)

**Table 7.25 Using <xsl:call-template> element**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          version="1.0">
  <xsl:template name="Choose_Brand" match="/CAMERAS/CAMERA">
    <xsl:value-of select="BRAND"/>
  </xsl:template>
  <xsl:call-template name="Choose_Brand"/>
</xsl:stylesheet>
```

In table 7.25, the XSL template, named Choose_Brand, is called using <xsl:call-template> element. The content of the <xsl:call-template> element may contain zero or more with-param elements.

### 7.6.20 Using Parameters

During XSLT transformation, it can be needed to save and reuse of some values. Value saving and reusing process is made by using variables and constants in traditional programming languages like Delphi, C++. There are two methods of working with XSL variables (WEB_16, 2004);

- By using <xsl:param> element
- By using <xsl:variable> element

### 7.6.21 Using <xsl:param> Element To Work With Parameters

Like functions, template rules can be invoked with parameters. To support parameters, the template rule's may begin with one or more <xsl:param> element that declare a named parameter and set its default value. All template parameters are added to the context of the template and are available to XPath expressions by prefixing the parameter name with a $. <xsl:param> has two attributes (WEB_16, 2004);

- ***name***, is the name of the parameter that is used to separate each parameter from other.
- ***select***, assigns values to parameters..

A parameter declaration can use either an XPath expression or a template as its children to set the default value of the parameter.

**Table 7.26 Using <xsl:param> element**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           version="1.0">
  <xsl:template match="/">
   <xsl:param name="P1">First Parameter</xsl:param>
   <xsl:param name="P2">Second Parameter</xsl:param>
   <xsl:if test="$P1='First Parameter'">
     <P>First parameter is chosen</P>
   </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

In table 7.26, P1 and P2 parameters are declared. Each parameter's values are assigned manually between <xsl:param>-</xsl:param> starting and closing elements. To reach the value of defined parameter, $ character is added into the beginning of the parameter name.

*$P1='First Parameter'* conditional declaration is used to control the value of the parameter named P1. Since the conditional definer's result is true, *<P>First parameter is chosen</P>* expression is added to output document.

The other way to reach an XSLT parameter's value is to locate parameter names within curly brackets in addition to $ character. This method is usually accepted when assigning values to the XSLT output elements' attributes (Harold E.R., 1999).

**Table 7.27 Using <xsl:param> element in attribute values**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
  <xsl:template match="/table">
    <xsl:param name="background"/>
    <table bgcolor="{$background}">
    </table>
  </xsl:template>
</xsl:stylesheet>
```

In table 7.27, background parameter is declared in the table XSLT template. This parameter is used to assign <table> element's bgcolor attribute in the HTML output that is generated after XSLT transformation.

**Table 7.28 Using <xsl:param> element's select attribute**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
  <xsl:template match="/table">
    <xsl:param name="background" select="’Blue’"/>
    <xsl:param name="border" select="3"/>
    <xsl:param name="alignment" select="’left’"/>
    <table bgcolor="{$background}" border="{$border}">
      <tr>
        <td align="{$alignment}">Usage of select attribute...</td>
      </tr>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

In table 7.28, three parameters with the names of *background*, *border* and *alignment* are defined. Parameter values are assigned with the usage of *select* attribute that assigns a default value for the parameter. Two of the values *left* and *blue* are established in single quote characters. String values that are wished to be assigned as default values must be settled in single quotes. Otherwise XSLT processor supposes that as an XML element.

The other way to assign default values to parameters is to add values between <xsl:param>-</xsl:param> elements (WEB_16, 2004). The values can be inserted directly or between <xsl:text> elements. So there is no more needed to use single quotes when declaring values.

**Table 7.29 Using parameters**

**[xsl-param.xml]**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<?xml-stylesheet type="text/xsl" href="xsl-param.xsl"?>
<table>
  <Param>
    <Screen>1024x768</Screen>
  </Param>
</table>
```

**[xsl-param.xsl]**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              version="1.0">
  <xsl:template match="/table">
    <xsl:param name="prmScreen" select="Param/Screen"/>
    <xsl:param name="table-width">
     <xsl:choose>
      <xsl:when test="$prmScreen='800x600'">
        <xsl:text>80</xsl:text>
      </xsl:when>
      <xsl:when test="$prmScreen='1024x768'">
        <xsl:text>120</xsl:text>
      </xsl:when>
     </xsl:choose>
    </xsl:param>
    <xsl:param name="image-width" select="$prmScreen - 20"/>
    <HTML>
      <BODY>
        <TABLE width="{$table-width}" border="1">
          <TR><TD><img width="{$image-width}" src="logo.gif"/></TD></TR>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

In table 7.29, default parameter value assigning and usage technique is represented during XSLT transformation. Screen element's value is attached to prmScreen parameter's value and *table-width* parameter's value is attached by using prmScreen parameter's value in a conditional define. If *image-width*'s value is *800x600* then *1024x768* is assigned or if *image-width*'s value is *1024x768* then *120* is assigned.

### 7.6.22 Passing Parameters To Templates

The <with-param> element is used within either the <apply-templates> or <call-template> element to set the value of local parameters used inside a template. If the select attribute is supplied, this element should be empty. Otherwise, the body contains the value to be assigned to the parameter. (WEB_16, 2004)

**Table 7.30 Passing parameters to templates**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          version="1.0">
 <xsl:template name="tmpTemplate" match="/">
  <xsl:param name="prmP1" select="/Params/P1">
  <xsl:choose>
   <xsl:when test="$P1='9811'">
    <P>9811</P>
   </xsl:when>
   <xsl:when test="$P1='9812'">
Table 7.30 Continued...
    <P>9812</P>
   </xsl:when>
  </xsl:choose>
 </xsl:template>
 <xsl:call-template name="tmpTemplate">
  <xsl:with-param name="prmP1" select="9811"/>
  </xsl:call-template>
</xsl:stylesheet>
```

In table 7.30, the parameter named *tmpTemplate* is called and a value is passed to *prmP1* parameter. The value of the passed parameter is 9811 and the output document is transformed as *<P>9811</P>*.

### 7.6.23 General Parameters

Template parameters are called local parameters and they can be only hold and used in that template's structure. Whereas, general parameters can be reached from anywhere in the XSLT document. (Harold E.R., 1999) Declaring technique of general parameters are totally same as local parameters. Only declaration locations of general parameters are different and they are defined at the same level of XSLT templates.

**Table 7.31 General parameter description**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          version="1.0">
 <xsl:param name="prmP1" select="/Params/P1"/>
 <xsl:param name="prmP2" select="'4566'"/>
 <xsl:template name="tmpTemplate" match="/">
  <xsl:choose>
   <xsl:when test="$prmP1='9811'">
    <P>9811</P>
   </xsl:when>
   <xsl:when test="$prmP2='4566'">
    <P>4566</P>
   </xsl:when>
  </xsl:choose>
 </xsl:template>
</xsl:stylesheet>
```

In 7.31, two general parameters with the name of *prmP1* and *prmP2* are declared. Parameters are located with the same level of templates (<xsl:template>) and under the <xsl:stylesheet> XSLT root element.

### 7.6.24 Using <xsl:variable> Element To Work With Variables

The xsl:variable element defines a named string for use elsewhere in the style sheet via an attribute value template. The xsl:variable element is an empty element that appears as a direct child of xsl:stylesheet . It has a single attribute name, which provides a name by which the variable can be referred to. The con-tents of the xsl:variable element provides the replacement text. <xsl:variable> element has two attributes like <xsl:param> element (WEB_16, 2004);

- ***name***, is the name of the variable that is used to separate each variable from other.
- ***select***, assigns values to variables..

**Table 7.32 Using <xsl:variable> element**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
  <xsl:template match="/">
    <xsl:variable name="V1">First Variable</xsl:variable>
    <xsl:if test="$V1='First Variable'">
      <P>First Variable is chosen</P>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

In table 7.32, a variable with name of *V1* is declared. *First Variable* value is assigned to variable as default. The value of the variable is appointed between <xsl:variable> and </xsl: variable> elements. $ character is inserted beginning of the variable name to reference the variable. *$V1='First Variable* conditional define is used to control the variable's value.

### 7.6.25 General Variables

Like in parameters, declaring technique of general variables is very same as local variables. Template variables are called local variables and they can be only hold and used in that template's structure. Whereas, general variables can be reached from anywhere in the XSLT document. Only declaration locations of general variables are different and they are defined at the same level of XSLT templates. If local and general variables are defined with the same name, local variables have more priority. (Harold E.R., 1999)

**Table 7.33 General variables**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
  <xsl: variable name="V1" select="'4566'"/>
  <xsl:template name="tmpTemplate" match="/">
    <xsl:choose>
      <xsl:when test="$V1=4566'">
        <P>4566</P>
      </xsl:when>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

In table 7.33, general variable with the name of *V1* is defined. Variable is declared at the same level with templates and just under the XSLT root element.

### 7.6.26 Difference Between XSLT Parameters And Variables

Although XSLT variables and parameters technically resemble in many structural definition, some differences can not be undervalued. They can be ordered like below;

1. It is not possible to override the values of <xsl:variable> element by using <xsl:with-param> method.
2. <xsl:param> elements must be declared just under the XSLT templates. <xsl:variable> elements can be defined anywhere inside or outside of the templates.

(WEB_16, 2004)

### 7.6.27 Optimization Over Complex Documents - <xsl:key>

The <xsl:key> element declares a named key for use with the key() function in XML Path Language expressions for enabling easier access into complex XML documents. Element has three attributes (WEB_16, 2004);

- **name**, identifies the key within the XSL Transformations (XSLT) document.
- **match**, specifies the pattern that should be matched. The Extensible Stylesheet Language (XSL) processor effectively preprocesses the source document and identifies all elements within the document that match the given pattern.
- **use**, provides the value to reference an element that satisfies the pattern specified in the **match** attribute..

The <xsl:key> element is designed to be used with the XPath key() function. The key() function takes two arguments, the name of the key and its value and then either retrieves the node or nodes associated with that key-value pair, or returns an empty node-set if no matching element is found. (Harold E.R., 1999) Unlike id elements, a key() expression may be matched by more than one node.

**Table 7.34 Using <xsl:key> element**

**[xsl-key.xml]**

```
<?xml version="1.0"?>
<?xml-stylesheet href="key_sample.xsl" type="text/xsl"?>
<PERSONNELLIST>
 <PERSONNEL ID="0001" DEP="SOFTWARE">
  <NAME>SIBEL KOPARAN</NAME>
  <EMAIL>SIBELK@NETSIS.COM.TR</EMAIL>
 </PERSONNEL>
 <PERSONNEL ID="0002" DEP="SOFTWARE">
  <NAME>DENIZ KILINC</NAME>
  <EMAIL>DENIZ.KILINC@NETSIS.COM.TR</EMAIL>
 </PERSONNEL>
 <PERSONNEL ID="0003" DEP="SUPPORT">
  <NAME>ALI ASKAN</NAME>
  <EMAIL>AASKAN@NETSIS.COM.TR</EMAIL>
 </PERSONNEL>
</PERSONNELLIST>
```

**[key_sample.xsl]**

**Table 7.34 Continued...**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
 <xsl:key name="findDepartment"
          match="PERSONNELLIST/PERSONNEL" use="@DEP"/>
 <xsl:template match="/">
  <HTML>
   <BODY>
    <TABLE BORDER="1">
    <xsl:for-each select="key('findDepartment', 'SOFTWARE')">
     <TR>
      <TD><xsl:value-of select="NAME"/></TD>
     </TR>
    </xsl:for-each>
    </TABLE>
   </BODY>
  </HTML>
 </xsl:template>
</xsl:stylesheet>
```

In table 7.34, an XSLT key with the name of *findDepartment* is created that chooses PERSONNEL element under the PERSONNELLIST root element and uses PERSONNEL element's DEP attribute as the key value. *key('findDepartment', 'SOFTWARE')* XPath expression provides to find elements which have the *SOFTWARE* value in their *findDepartment* key.

### 7.6.28 Ordering XML Elements - <xsl:sort>

The <xsl:sort> element sorts the output elements into a different order than they appear in the input (WEB_16, 2004). An <xsl:sort> element appears as a child of an <xsl:apply- templates> element or <xsl:for-each> element. The *select* attribute of the <xsl:sort> element defines the key used to sort the elements' output by <xsl:apply-templates> or <xsl:for-each>. <xsl:sort> elements can be considered as SQL (Simple Query Language) *ORDER-BY clause* over XML data sources. Element has 5 attributes (Harold E.R., 1999);

- **select**, Specifies a sort key for the node. The default value of the select attribute is " . ".
- **data-type**, Specifies the data type of the strings. If no data type is specified, the type of the expression will be used as the default.
- **order**, Specifies whether the strings will be sorted in ascending or descending order. The default is ascending.
- **lang**, Specifies which language's alphabet is used to determine sort order. If no lang value is specified, the language is determined from the system environment.
- **case-order**, Specifies whether the strings will be sorted with lower-case or uppercase characters specified first. The default is upper-first.,.

If more than one xsl:sort element is present in a given xsl:apply-templates or xsl:for-each element, then the output sorts first by the first key, then by the second key, and so on.

**Table 7.35 Ordering elements**

| **[xsl-sort.xsl]** |
|---|
| <pre><?xml version="1.0"?><br><xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"<br>            version="1.0"><br> <xsl:template match="PERSONNELLIST"><br>  <TABLE border="1"><br>   <xsl:for-each select="PERSONNEL"><br>     <xsl:sort select="@DEP" order="ascending"/><br>     <xsl:sort select="NAME" order="descending"/><br>     <TR><br>       <TD><xsl:value-of select="@DEP"/></TD><br>       <TD><xsl:value-of select="NAME"/></TD><br>     </TR><br>   </xsl:for-each><br>  </TABLE><br></xsl:template><br></xsl:stylesheet></pre> |
| **[Output]** |

**Table 7.35 Continued...**
```
<TABLE border="1">
  <TR><TD>SUPPORT  </TD><TD>ALI ASKAN</TD></TR>
  <TR><TD>SOFTWARE</TD><TD> SIBEL KOPARAN</TD></TR>
  <TR><TD>SOFTWARE</TD><TD> DENIZ KILINC    </TD></TR>
</TABLE >
```

In table 7.35, an XSL document with the name of *xsl-sort.xsl* is showed that chooses, orders and transforms personnel, department and name information in the *xsl-key.xml* XML document. Document content includes two order keys. The first key orders according to each personnel's department information and the second one orders according to each personnel's name information.

Key elements are processed in order to their physical locations in the XSLT document as primary and secondary keys. Therefore, the key over the department information is primary and first ordering is constituted over it. The result of the first ordering is below;

**Table 7.36 Result according to first ordering key**

| DEP | NAME |
|---|---|
| **SOFTWARE** | DENIZ KILINC |
| **SOFTWARE** | SIBEL KOPARAN |
| **SUPPORT** | ALI ASKAN |

Since primary key projects an *ascending* and alphabetical ordering over department information, first ordering result is realized as above. Because alphabetically *SOFTWARE* word is comes before from the *SUPPORT* word. As a result personals at the *SOFTWARE* department are local in the most top position.

After the first key processing, second key is activated. Second key projects an *descending* and alphabetical ordering over name information. Personnel elements that have been ordered according to primary key, are now ordered according to their names. The result of the ordering is below. In this table DENIZ and SIBEL's places are changed each of them works in SOFTWARE department. The reason of this difference is the first character of *SIBEL* word *S* comes after first character of *DENIZ* word *D* in alphabet.

**Table 7.37 Result according to first and second ordering key**

| DEP | NAME |
|---|---|
| **SOFTWARE** | SIBEL KOPARAN |
| **SOFTWARE** | DENIZ KILINC |
| **SUPPORT** | ALI ASKAN |

### 7.6.29 Converting Values To String Data – string()

String() XPath function is used during the XSLT transformation to convert values to string data. Especially, this function is necessary while adding the result of arithmetic calculations. (WEB_16, 2004) String() function can convert XSLT parameters and variables to string literals.

**Table 7.38 Using string() function**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
 <xsl:template name="trapezoidAreaCalculation">
  <xsl:param name="prmLowerFloor"/>
  <xsl:param name="prmTopFloor"/>
  <xsl:param name="prmHighness"/>
  <xsl:value-of select="String(($prmLowerFloor+$prmTopFloor)
                              * (prmHighness/2))"/>
 </xsl:template>
 <xsl:call-template name="trapezoidAreaCalculation ">
  <xsl:with-param name="prmLowerFloor" select="5"/>
  <xsl:with-param name="prmTopFloor" select="4"/>
  <xsl:with-param name="prmHighness" select="6"/>
 </xsl:call-template>
</xsl:stylesheet>
```

In table 7.38, a template with the name of *trapezoidAreaCalculation* is declared that calculates the area of geometric *trapezoid* shape. Template takes three parameters, calculates the area and inserts the result to output document. Area calculation is a mathematical operation. XPath string() function has used to insert of the operation result into the output document.

### 7.6.30 Working With White-Spaces

XSL Transformations (XSLT) can distinguish nodes that contain white space intermingled with other characters. The white space is considered inseparable from the other text in the node. For nodes that contain nothing but white space, the <xsl:preserve-space> and <xsl:strip-space> elements handle how the nodes are output.

1. **<xsl:preserve-space>** is a white space-preserving element and has an *elements* attribute whose value is a white space separated list of name tests (WEB_16, 2004). Initially, the set of white space-preserving element names contains all element names. *<xsl:preserve-space elements="NAME PHONE"/>.NAME* and *PHONE* are the names of all elements whose white space must be preserved. It can be specified all of the elements in the source document with the asterisk operator. Because all of the content of an XML document is, by default, preserved, <xsl:preserve-space> is useful only in cases in which <xsl:strip-space> element have used (Harold E.R., 1999).
2. **<xsl:strip-space>** element provides a list of those elements in the source document in which content must be removed from the output tree. <xsl:strip-space> is an empty, top-level element. (WEB_16, 2004) *<xsl:strip-space elements="NAME PHONE"/>*. *NAME* and *PHONE* are the names of all elements whose white space must be removed.

If an element appears in both an <xsl:strip-space> and <xsl:preserve-space> list, the last specification applies which is an either <xsl:strip-space> element or <xsl:preserve-space> element. (WEB_16, 2004)

**Table 7.39 Working with white-spaces**

**[preserve-strip.xml]**

```
<?xml-stylesheet type="text/xsl" href="preserve-strip.xsl" ?>
<SPACETEST>
 <exclude> </exclude>
 <include> </include>
</SPACETEST>
```

**[preserve-strip.xsl]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:preserve-space elements="include"/>
<xsl:strip-space elements="exclude"/>
<xsl:template match="SPACETEST">
 <xsl:for-each select="exclude">
  Exclude: [<xsl:value-of select="translate(.,' ','*')"/>]
 </xsl:for-each>
 <xsl:for-each select="include">
  Include: [<xsl:value-of select="translate(.,' ','*')"/>]
 </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
Exclude: []
```

**Table 7.39Continued...**

```
Include: [*]
```

In table 7.39, *preserve-strip.xml* XML document that contains <exclude> and <include> elements, *preserve-strip.xsl* XSL document which processes white-spaces in the XML document. White space characters are converted to asterisk operator (*) by using the XSLT translate function. <include> element's white space is preserved and <exclude> element's white space is stripped according to declarations in XSL document.

### 7.6.31 Normalizing White-Spaces

XSL Transformations (XSLT) provides a built-in function, normalize-space(), that strips leading and trailing white space from a string and normalizes multiple successive white space characters to a single space. It takes one argument, a string or node-set, if the argument is omitted, normalize-space() assumes the context node. (WEB_16, 2004)

**Table 7.40 Working with white-spaces**

**[normalize.xml]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<?xml-stylesheet type="text/xsl" href="normalize.xsl" ?>
<SPACENORMALIZE>
  <VALUE>   How spaces
   are              normalized?
 </VALUE>
</SPACENORMALIZE>
```

**[normalize.xsl]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="SPACENORMALIZE">
   Value: [<xsl:value-of select="normalize-space(VALUE)"/>]
</xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
Value: [How spaces are normalized?]
```

In table 7.40, an XML document that contains white-spaces, an XSL document that realizes transformation process and normalizes white-spaces and the output document are showed. XPath normalize-space() function strips leading and trailing white space from a string and normalizes multiple spaces to single character in the XML *VALUE* element.

### 7.6.32 XSLT String Functions

XSLT technology supports many XPath functions to work sting literals during XSLT transformation. Functions are listed below;

**Table 7.41 XSLT string functions (WEB_16, 2004)**

| Funcion | Description | Prototype |
|---|---|---|
| Concat | Concats more than one string values. | concat(value,value,value*) |
| contains | Controls whether or not a value contains another value. Function returns boolean *true* or *false* value. | contains(value,value) |
| string-length | Returns the number of characters in a value. | string-length(value) |
| starts-with | Controls whether or not a value starts with another value. Function returns boolean *true* or *false* value. | starts-with(deger,deger) |
| substring | Copies a value's sub-part from the specified character starting location. | substring(value,digit,digit) |
| substring-before | Copies a value's sub-part before the specified value. | substring-before(value,value) |
| substring-after | Copies a value's sub-part after the specified value. | substring-after(value,value) |
| Translate | Provides to convert characters of some values to other characters. | translate(value,value,value) |

### 7.6.33 Joining String Values – concat()

XPath *concat()* function is used to join more than one string values into the one string value. In addition to string values, function may take XSLT parameters and variables as input.

**Table 7.42 Usage of concat() function**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
 <xsl:template match="/">
  <xsl:param name="prmHead">HEAD_</xsl:param>
```

**Table 7.42 Continued...**

```
  <xsl:variable name="varBody">BODY</xsl:variable>
  <xsl:value-of select="concat($prmHead,$varBody)"/>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
HEAD_BODY
```

In table 7.42, XPath concat function takes an XSLT parameter and variable as input parameters. The input values are added from left to right and as a result a new value is constituted.

## 7.6.34 Controlling Containing String Values – contains()

XPath contains function is used to control whether or not a value contains another value. Function takes two parameters. First parameter is the source value and the second parameter is the searched value. If the first argument string contains the second argument string, function returns *true* otherwise *false (WEB_16, 2004)*.

**Table 7.43 Usage of contains() function**

**[contains.xml]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<?xml-stylesheet type="text/xsl" href="contains.xsl" ?>
<STOCK_MASTER_INFO>
 <STOCK CODE="S001">
  <SNAME>VR1-ONE CHANNEL RADIO CARD</SNAME>
  <SAILING_PRICE>9000000</SAILING_PRICE>
  <BUYING_PRICE>8800000</BUYING_PRICE>
  <VAT>18</VAT>
  <GRUP_CODE>RADIO</GRUP_CODE>
 </STOCK>
 <STOCK CODE="S002">
  <SNAME>TO3- THREE CHANNEL REMOTE CONTROLLER</SNAME>
  <SAILING_PRICE>8000000</SAILING_PRICE>
```

**Table 7.43 Continued...**

```
   <BUYING_PRICE>7000000</BUYING_PRICE>
   <VAT>15</VAT>
   <GRUP_CODE>REMOTE CONTROLLER</GRUP_CODE>
  </STOCK>
</STOCK_MASTER_INFO>
```

**[contains.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
             version="1.0">
 <xsl:template match="STOCK_MASTER_INFO">
   <xsl:for-each select="STOCK">
     <xsl:if test="contains(normalize-space(SNAME),'CHANNEL')">
       <xsl:value-of select="SNAME"/><BR/>
     </xsl:if>
   </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
VR1-ONE CHANNEL RADIO CARD
TO3- THREE CHANNEL REMOTE CONTROLLER
```

In table 7.43, an XML document that contains stock master information, an XSLT document that realizes transformation process by using contains function and output document are showed. *contains* function takes stock master information's SNAME element as the source argument and *CHANNEL* keyword as the second string argument that will be searched. This query controls whether or not stock names have *CHANNEL* string. *normalize-space* function is used to strip spaces in the stock names.

### 7.6.35 Finding Number of Characters in Strings – string-length()

string-length() function is used to calculate the number of the characters in string literals. Function takes a string value as an input argument. If the argument is omitted, it returns the number of characters in the string-value of the context node. If an argument is not of type *string,* it is first converted to a string and then evaluated (WEB_16, 2004).

**Table 7.44 Usage of string-length() function**

**[string-length.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          version="1.0">
 <xsl:template match="STOCK_MASTER_INFO">
  <xsl:for-each select="STOCK">
   <xsl:text>SNAME:</xsl:text>
   <xsl:value-of select="string-length(SNAME)"/>,
   <xsl:text>GRUP_CODE:</xsl:text>
   <xsl:value-of select="string-length(GRUP_CODE)"/><BR/>
  </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
SNAME:31, GRUP_CODE:5
SNAME:29, GRUP_CODE:7
```

In table 7.44, an XSLT document with the name of *string-length.xsl* and the output document is showed. string-length function is used to calculate the number of characters in *SNAME* and *GRUP_CODE* elements. As a result, numeric values are inserted to output document.

### 7.6.36 Comparing Starting Substrings - starts-with()

XPath starts-with function is used to find whether or not a value starts with another value. Function takes two parameters. First parameter is the source value and the second parameter is the searched value. It returns *true* if the first argument string starts with the second argument string; otherwise returns *false*. If an argument is not of type *string,* it is first converted to a string and then evaluated. (WEB_16, 2004)

**Table 7.45 Usage of starts-with() function**

**[starts-with.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          version="1.0">
 <xsl:template match="STOCK_MASTER_INFO">
  <xsl:for-each select="STOCK">
```

**Table 7.45 Continued...**

```
   <xsl:if test="starts-with(GRUP_CODE,'K')">
     SNAME: <xsl:value-of select="SNAME"/>
     GRUP_CODE: <xsl:value-of select="GRUP_CODE"/>
   </xsl:if>
  </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
TO3- THREE CHANNEL REMOTE CONTROLLER
GRUP_CODE: REMOTE CONTROLLER
```

In table 7.45, an XSLT document that processes and transforms *contains.xml* XML document and output document is showed. XPath *starts-with* function is used to find whether or not *GRUP_CODE* elements start with alphabetical *K* character. Result values are inserted to output document.

### 7.6.37 Choosing Substrings - substring()

Returns the substring of the first argument starting at the position specified in the second argument and the length specified in the third argument. Each character in the string is considered to have a numeric position (WEB_16, 2004): the position of the first character is 1, the position of the second character is 2, and so on. If the third argument is not specified, it returns the substring starting at the position specified in the second argument and continuing to the end of the string.

**Table 7.46 Usage of substring() function**

**[substring.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
 <xsl:template match="STOCK_MASTER_INFO">
  <xsl:for-each select="STOCK">
   <xsl:value-of select="substring(SNAME,1,10)"/><BR/>
  </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

**Table 7.46 Continued...**

**[Output]**

```
VR1-ONE CH
TO3-THREE
```

In table 7.46, an XSLT document that processes and transforms *contains.xml* XML document and output document of the transformation is showed. SNAME elements are selected from their 1st characters to their 10th character and copied into the output document by using *substring* function.

### 7.6.38 Choosing Substrings Before Substrings – substring-before()

*substring-before* function is used to select a string' sub part before the specified position during XSLT transformation. It returns the substring of the first argument string that precedes the first occurrence of the second argument string in the first argument string, or the empty string if the first argument string does not contain the second argument string. (WEB_16, 2004)

**Table 7.47 Usage of substring-before() function**

**[substring-before.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">
 <xsl:template match="STOCK_MASTER_INFO">
  <xsl:for-each select="STOCK">
   <xsl:value-of select="substring-before(SNAME,'-')"/><BR/>
  </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
VR1
TO3
```

In table 7.47, an XSLT document with the name of *substring-before.xsl* that processes and transforms *contains.xml* XML document and output document of the transformation are showed. SNAME elements' sub parts are copied by selecting the characters before "-" character by using XPath *substring-before* function.

### 7.6.39 Choosing Substrings After Substrings – substring-after()

*substring-after* function is used to select a string' sub part after the specified position during XSLT transformation. It returns the substring of the first argument string that follows the first occurrence of the second argument string in the first argument string, or the empty string if the first argument string does not contain the second argument string. (WEB_16, 2004)

**Table 7.48 Usage of substring-after() function**

**[substring-after.xsl]**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          version="1.0">
 <xsl:template match="STOCK_MASTER_INFO">
  <xsl:for-each select="STOCK">
   <xsl:value-of select="substring-after(SNAME,'-')"/><BR/>
  </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
ONE CHANNEL RADIO CARD
THREE CHANNEL REMOTE CONTROLLER
```

In table 7.48, an XSLT document with the name of *substring-before.xsl* that processes and transforms *contains.xml* XML document and output document of the transformation are showed. SNAME elements' sub parts are copied by selecting the characters after "-" character by using XPath *substring-after* function.

### 7.6.40 Replacing String Characters – translate()

*translate* XPath function is used to replace some specified characters with others. First parameter is the source value that is also the main string literal. Second parameter is the string value that will be replaces with new values. And the third parameter contains new values which will be replaced. (WEB_16, 2004)

**Table 7.49 Usage of translate() function**

**[translate.xml]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<?xml-stylesheet type="text/xsl" href="translate.xsl" ?>
<ENCRYPTED_VALUE>
 Deniz KILINÇ
</ENCRYPTED_VALUE>
```

**[translate.xsl]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                 version="1.0">
 <xsl:template match="/">
   <B>Encrypted Value: </B>
<xsl:value-of select="translate
(ENCRYPTED_VALUE,'ABCDEHIİKLMN','12345€@#$½._')"/><BR/>
   <B>Decrypted Value: </B><xsl:value-of select="translate
(ENCRYPTED_VALUE,'12345€@#$½._','ABCDEHIİKLMN')"/><BR/>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

**Encrypted Value:** 4eniz $@½@_Ç
**Decrypted Value:** Deniz KILINÇ

In table 7.49, an xml document with the name of *translate.xml* that includes the element named *ENCRYPTED_VALUE*, an XSLT document which will transform XML document and output document are showed. *Translate* function is used for encryption and decryption processes. According to encryption logic *ABCDEHI İKLMN* source characters will be replaced with new characters *12345€@#$½._*. And decryption is realized in the opposite manner.

### 7.6.41 Converting Values To Numeric Data – number()

number() XPath function is used during the XSLT transformation to convert values to numeric data. In addition to normal values, number() function can also convert XSLT parameters and variables to numeric values. The type of the function parameters are below (WEB_16, 2004);

- String values that match the IEEE 754 standard (the others are called and processed as NaN).
- Node values which are converted IEEE 754 string characters by using string() function.
- Boolean values, (True value is converted to 1 digit and false value is converted to 0 digit).

**Table 7.50 Using number() function**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          version="1.0">
 <xsl:template match="/">
  <xsl:value-of select="number('a')"/><BR/>
  <xsl:value-of select="number('a'*4)"/><BR/>
  <xsl:value-of select="number(4*4)*4*number(true())"/>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
NaN
NaN
64
```

In table 7.50, three usage type of XPath *number()* function is showed. In the first usage, since the string character *a* does not match IEEE 754 standart, NaN is inserted into output document. In the second usage, *'a'*4* expression is not processed with the same reason and NaN is again inserted into output document. In the third usage, *4*4* expression is converted numeric value and then calculated with the number 4. The result 64 is finally calculated with *number(true())* expression that is equal to 1 digit.

### 7.6.42 XSLT Number Functions

XSLT technology supports many XPath functions to work numbers during XSLT transformation. Functions are listed below;

**Table 7.51 XSLT number functions (WEB_16, 2004)**

| Function | Description | Prototype |
|---|---|---|
| Round | It returns an integer closest in value to the argument. | round(value) |
| Ceiling | It returns the smallest integer that is not less than the argument. | ceiling(value) |
| Floor | It returns the largest integer that is not greater than the argument. | floor(value) |
| Sum | It returns the sum of all nodes in the node set. | sum(node-set) |

### 7.6.43 XSLT round() Function

Round() function is used to convert numeric values to closest integer values to them. Function may take parameters as below (WEB_16, 2004);

- Positive parameter values are returned as positive.
- Negative parameter values are returned as negative.
- Values between 0 and -0.5 are returned as 0 value.

**Table 7.52 Using round() function**

| | | |
|---|---|---|
| <xsl:value-of select="round(0.4)"/><BR/> | → | 0 |
| <xsl:value-of select="round(1.4)"/><BR/> | → | 1 |
| <xsl:value-of select="round(1.5)"/><BR/> | → | 2 |
| <xsl:value-of select="round(-0.2)"/><BR/> | → | 0 |
| <xsl:value-of select="round(-0.6)"/><BR/> | → | -1 |

In table 7.52, round() function's usages are sampled. The values are always converted to closest values to them. Although the value 1.5 has same distance to 1 and 2 values, it is converted to the 2 value. In round function, these types of values are converted to upper closest values.

### 7.6.44 XSLT Ceiling() Function

Ceiling function is used to convert numeric values to biggest and closest integer that is not less than the value itself. (WEB_16, 2004)

**Table 7.53 Using ceiling() function**

| | | |
|---|---|---|
| <xsl:value-of select="ceiling(0.0001)"/><BR/> | → | 1 |
| <xsl:value-of select="ceiling(1.4)"/><BR/> | → | 2 |
| <xsl:value-of select="ceiling(1.5)"/><BR/> | → | 2 |
| <xsl:value-of select="ceiling(-1.2)"/><BR/> | → | -1 |

In table 7.53, some usage techniques of *ceiling* function are showed. Values are always converted to numeric values to biggest and closest integer to themselves. If it is looked carefully *0.0001* numeric value is converted to 1 instead of 0.

### 7.6.45 XSLT Floor() Function

Floor function is used to convert numeric values to smallest and closest integer that is not bigger than the value itself. (WEB_16, 2004)

**Table 7.54 Using floor() function**

| | | |
|---|---|---|
| <xsl:value-of select="ceiling(0.99999)"/><BR/> | → | 0 |
| <xsl:value-of select="ceiling(1.2)"/><BR/> | → | 1 |
| <xsl:value-of select="ceiling(1.9)"/><BR/> | → | 1 |
| <xsl:value-of select="ceiling(-1.2)"/><BR/> | → | -2 |

In table 7.54, some usage techniques of *floor* function are showed. Values are always converted to numeric values to smallest and closest integer to themselves. For example; *0.99999* numeric value is converted to 0 instead of 1.

### 7.6.46 XSLT Sum() Function

Sum function is used to sum of all nodes in the node set. Each node is first converted to a number value before summing. (WEB_16, 2004)

**Table 7.55 Using sum() function**

**[sum.xml]**

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<?xml-stylesheet type="text/xsl" href="sum.xsl"?>
<ORDER_LIST>
 <ORDER CUSTOMER="001">
  <STOCK>STK-001</STOCK>
  <QUANTITY>4</QUANTITY>
 </ORDER>
 <ORDER CUSTOMER="001">
  <STOCK>STK-002</STOCK>
  <QUANTITY>6</QUANTITY>
 </ORDER>
 <ORDER CUSTOMER="002">
  <STOCK>STK-001</STOCK>
  <QUANTITY>4</QUANTITY>
 </ORDER>
</ORDER_LIST>
```

**[sum.xsl]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
         version="1.0">
 <xsl:template match="/">
  <TABLE>
   <TR bgcolor="silver">
    <TD>Sum of 001 customer's order </TD>
    <TD><B><xsl:value-of select="
sum(/ORDER_LIST/ORDER[@CUSTOMER='001']/QUANTITY)"/></B></TD>
   </TR>
   <TR bgcolor="silver">
    <TD>Sum of STK-001 stock's order quantity</TD>
    <TD>
<B><xsl:value-of select="
sum(/ORDER_LIST/ORDER[STOCK='STK-001']/QUANTITY)"/></B></TD>
   </TR>
```

**Table 7.55 Continued...**

```
  </TABLE>
 </xsl:template>
</xsl:stylesheet>
```

**[Output]**

```
Sum of 001 customer's order              10
Sum of STK-001 stock's order quantity    8
```

In table 7.55, an XML document with the name of *sum.xml* that contains an order list, an XSL document with the name of *sum.xsl* that processes and transforms XML document and output HTML document are showed. XSLT sum function is used to calculate the sum of order quantities for customer and stock based.

- The XPath expression at below is used to calculate sum of the order quantity of customer with the code of 001;
  *sum(/ORDER_LIST/ORDER[@CUSTOMER='001']/QUANTITY)*
- The XPath expression at below is used to calculate sum of the order quantity of stock with the code of STK-001;
  *sum(/ORDER_LIST/ORDER[STOCK='STK-001']/QUANTITY)*

### 7.6.47 XSLT decimal-format Function

The <decimal-format> element is a top-level element used to control the way the XPath format-number()function formats numbers into strings (WEB_16, 2004). This element neither affects the way the number and value-of elements format numbers for output nor the XPath string() function. The attributes of <decimal-format> XSLT element are listed below;

**Table 7.56 Attributes of <decimal-format> element (WEB_16, 2004)**

| Attribute Name | Description | Obligatory |
|---|---|---|
| decimal-separator | Specifies the character used to separate the integral whole number from its fractional counterpart. The default is ".". | Optional |

| Table 7.56 Continued... | | |
|---|---|---|
| Digit | Specifies the character used to indicate digits in the format pattern. The default is #. | Optional |
| grouping-separator | Specifies the character used to separate numbers exceeding thousands, millions, and so forth. The default is ,. | Optional |
| minus-sign | Specifies the character used to indicate negative numbers. The default is -. | Optional |
| Name | Binds a name to this decimal format. If omitted, this format becomes the default. | Optional |
| NaN (Not-a-Number) | Specifies the string used to represent a nonnumerical value, NaN, or "not-a-number." The default is NaN. | Optional |
| Percent | Specifies the character used to represent the percent sign. The default is %. | Optional |
| zero-digit | Specifies the character used in the format pattern to indicate leading zeroes. The default is 0. | Optional |
| Infinity | Specifies the string that represents a value of infinity. The default is Infinity. | Optional |
| per-mile | Specifies the character used to represent the per-mille sign or the sign used to represent the percentage of thousandths. The default is the Unicode character #x2030, ‰. | Optional |
| pattern-separator | Specifies the character used to separate the pattern used for positive values from the pattern used for negative numbers. The default is ;. | Optional |

<decimal-format> element only declares a decimal format, which controls the interpretation of a format pattern used by the *format-number*. If there is a name attribute, then the element declares a named decimal-format; otherwise, it declares the default decimal-format. (Harold E.R., 1999)

### 7.6.48 XSLT format-number Function

Format-number function converts numbers into strings. The conversion is the result of formatting the number specified in the first argument, using the format specified in the second argument, and applying the rules defined in the decimal format named in the third optional argument. (WEB_16, 2004)

If the third argument, the decimal format name, is omitted, the default decimal format is used. The third argument, represents the name of a decimal format, as specified using the <xsl:decimal-format> element. The decimal format name must be a qualified name. The second argument specifies the format pattern. For the "format pattern" string, the following characters are supported "."; ","; "#","%","0", and the Unicode per-mille character (#x2030). Characters and their functions are listed below. (WEB_16, 2004)

**Table 7.57 Pattern characters of format-number function (WEB_16, 2004)**

| Pattern Char | Description | Sample |
|---|---|---|
| # | It specifies the number of decimal steps. | The output of the 98112234 is 98,112,234 after the "#,###" pattern. |
| 0 | It specifies that the format will have the number 0. | The output of the 9811.5 is 9,811.50 after the "#,###.00" pattern. |
| . | It specifies the decimal dot character. | The output of the 9811.55 is 9811.55 after the "###.00" pattern. |
| - | It specifies the number is negative. | The output of the 11.55 is -11.55 after the "-###.00" pattern. |
| % | It specifies the number in percentage. | The output of the *35.52* is *%35.5* after the "%###.0" pattern. |
| , | It specifies grouping decimal separator. | - |
| ‰ (\u230) | It specifies the per-mille character. | - |

**Table 7.58 Usage of format-number function**

```
<xsl:decimal-format
      name="DecimalFormat"
      decimal-seperator="."
      grouping-seperator=","
      minus-sign="-"
```

| **Table 7.58 Continued...** |
|---|
| digit=”#”> <xsl:value-of select="format-number(3456, '#,##0.00', DecimalFormat)"/> |
| **Output** |
| 3,456 |

In table 7.58, <decimal-format> element’s usage is showed. Defined decimal format is passed to *format-number* XSLT function as the third argument. The formatted result of the 3456 number is 3,456.

### 7.6.49 Combining More Than One XSLT Documents

document() is one of the most useful function used in XSLT and XPath technologies. It lets the stylesheet name an additional document to read (WEB_16, 2004). Therefore, it will be easy to insert the whole document into the result tree or insert part of it based on a condition described by an XPath expression.

Accessed external documents can be managed using standard XSLT rules and XPath expressions. document() accepts a node-set as a parameter, in which case the function assumes each node is a URI (Uniform Resource Identifier). The function combines the two to decide which documents to load (WEB_16, 2004) Still, in most cases it is simpler to pass a single URI.Imagine a company that has different branch offices in different cities. This company also collects all order information of each branch in center. Order information is transferred to center in XML format. An audit report will be generated over the order information everyday in center office.

**Table 7.59 Usage of document function**

| **[center_ord.xml]** | **[izmir_ord.xml]** |
|---|---|
| <?xml version="1.0" encoding="ISO-8859-9"?> <ORDER_LIST> | <?xml version="1.0" encoding="ISO-8859-9"?> <ORDER_LIST> |

| **Table 7.59 Continued...** | |
|---|---|
| `<BRANCH>ANKARA</BRANCH>`<br>`<ORDER CUSTOMER="001">`<br>`  <STOCK>STK-001</STOCK>`<br>`<QUANTITY>4</QUANTITY>`<br>`  <PRICE>4000000</PRICE>`<br>`</ORDER>`<br>`<ORDER CUSTOMER="002">`<br>`  <STOCK>STK-002</STOCK>`<br>`  <QUANTITY>10</QUANTITY>`<br>`  <PRICE>8000000</PRICE>`<br>`</ORDER>`<br>`</ORDER_LIST>` | `<BRANCH>İZMİR</BRANCH>`<br>`<ORDER CUSTOMER="004">`<br>`  <STOCK>STK-004</STOCK>`<br>`  <QUANTITY>8</QUANTITY>`<br>`  <PRICE>8000000</PRICE>`<br>`</ORDER>`<br>`<ORDER CUSTOMER="005">`<br>`  <STOCK>STK-001</STOCK>`<br>`  <QUANTITY>10</QUANTITY>`<br>`  <PRICE>10000000</PRICE>`<br>`</ORDER>`<br>`</ORDER_LIST>` |

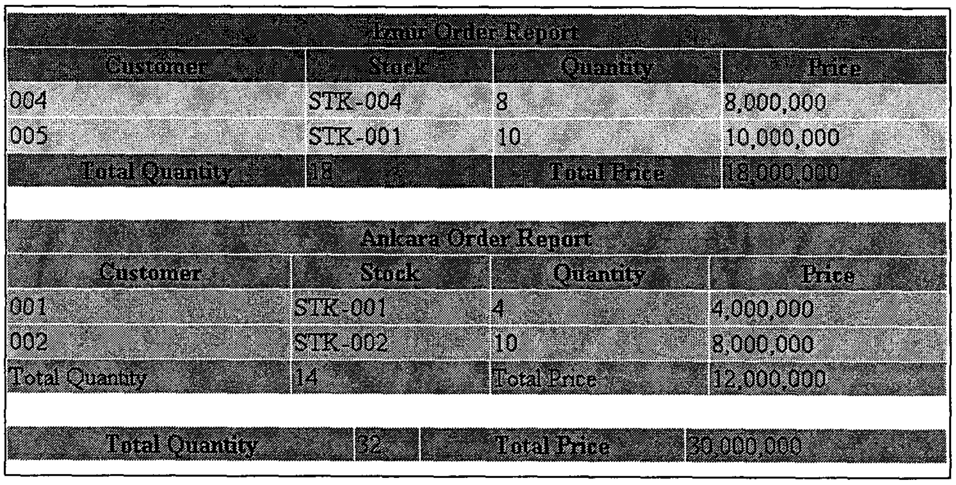
**[document.xsl]**

```
<?xml version="1.0" encoding="iso-8859-9"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          version="1.0">
 <xsl:template match="/">
   <xsl:variable name="quantitySumAnkara"
        select="sum(//ORDER_LIST/ORDER/QUANTITY)"/>
   <xsl:variable name="quantitySumIzmir"
select="sum(document('izmir_ord.xml')//ORDER_LIST/ORDER/QUANTITY)"/>
   <xsl:variable name="priceSumAnkara"
        select="sum(//ORDER_LIST/ORDER/PRICE)"/>
   <xsl:variable name="priceSumIzmir"
select="sum(document('izmir_ord.xml')//ORDER_LIST/ORDER/PRICE)"/>
   <TABLE width="60%" cellspacing="1" align="center">
     <TR bgcolor="dodgerblue">
     <TH COLSPAN="4">İzmir Order Report</TH></TR>
     <TR bgcolor="dodgerblue"><TH>Customer</TH><TH>Stock</TH>
     <TH>Quantity</TH><TH>Price</TH></TR>
     <xsl:for-each
          select="document('izmir_ord.xml')/ORDER_LIST/ORDER">
      <TR bgcolor="lightsteelblue">
       <TD><xsl:value-of select="@CUSTOMER"/></TD>
       <TD><xsl:value-of select="STOCK"/></TD>
       <TD><xsl:value-of select="QUANTITY"/></TD>
       <TD><xsl:value-of select="format-number(PRICE,'#,##0')"/></TD>
      </TR>
     </xsl:for-each>
     <TR bgcolor="dodgerblue"><TH>Total Quantity</TH>
      <TD><xsl:value-of select="$quantitySumIzmir"/></TD>
      <TH>Total Price</TH>
```

**Table 7.59 Continued...**

```
      <TD><xsl:value-of select="format-number($priceSumIzmir,'#,##0')"/></TD>
     </TR>
   </TABLE><BR/>
  <TABLE width="60%" cellspacing="1" align="center">
     <TR bgcolor="mediumseagreen">
     <TH COLSPAN="4">Ankara Order Report</TH></TR>
     <TR bgcolor="mediumseagreen"><TH>Customer</TH><TH>Stock</TH>
       <TH>Quantity</TH><TH>Price</TH></TR>
      <xsl:for-each select="ORDER_LIST/ORDER">
       <TR bgcolor="darkseagreen">
        <TD><xsl:value-of select="@CUSTOMER"/></TD>
        <TD><xsl:value-of select="STOCK"/></TD>
        <TD><xsl:value-of select="QUANTITY"/></TD>
        <TD><xsl:value-of select="format-number(PRICE,'#,##0')"/></TD>
       </TR>
     </xsl:for-each>
     <TR bgcolor="mediumseagreen">
      <TD>Total Quantity</TD>
      <TD><xsl:value-of select="$quantitySumAnkara"/></TD>
      <TD>Total Price</TD>
      <TD><xsl:value-of
                select="format-number($priceSumAnkara,'#,##0')"/></TD>
     </TR>
   </TABLE><BR/>
   <TABLE width="60%" cellspacing="1" align="center">
     <TR bgcolor="steelblue">
      <TH>Total Quantity</TH>
      <TD><xsl:value-of
            select="$quantitySumIzmir + $quantitySumAnkara"/></TD>
      <TH>Total Price</TH><TD><xsl:value-of
     select="format-number($priceSumAnkara + $priceSumIzmir,'#,##0')"/></TD>
     </TR>
   </TABLE>
  </xsl:template>
</xsl:stylesheet>
```

In table 7.59, XML documents that contain Ankara and İzmir branch offices' daily order information and an XSLT document with the name of *document.xsl* that combines branch office and center office XML and generates an audit report.



| İzmir Order Report | | | |
|---|---|---|---|
| Customer | Stock | Quantity | Price |
| 004 | STK-004 | 8 | 8,000,000 |
| 005 | STK-001 | 10 | 10,000,000 |
| Total Quantity | 18 | Total Price | 18,000,000 |

| Ankara Order Report | | | |
|---|---|---|---|
| Customer | Stock | Quantity | Price |
| 001 | STK-001 | 4 | 4,000,000 |
| 002 | STK-002 | 10 | 8,000,000 |
| Total Quantity | 14 | Total Price | 12,000,000 |

| | | | |
|---|---|---|---|
| Total Quantity | 32 | Total Price | 30,000,000 |

**Figure 7.5 Merging XSLT Documents**

XPath document() function is used to access İzmir branch's order information. Usage techniques are below;

- Expression is used to find the total amount of order of İzmir customers;
  *sum(**document**('izmir_ord.xml')//ORDER_LIST/ORDER/QUANTITY)*
- Expression is used to find the total price of İzmir customers;
  *sum(**document**('izmir_ord.xml')//ORDER_LIST/ORDER/PRICE)*
- Expression is used to process the İzmir customer's orders iteratively;
  *select="**document**('izmir_ord.xml')/ORDER_LIST/ORDER*

Center office's order information is processed and merged into the output document in addition to İzmir branch's order information. It is also provided to show all of the order quantities and prices by keeping the each office's total order quantity and price in a XSLT variable.

XPath document() function can also be used to traverse XSLT document itself by passing no parameters to function. Therefore, document function is necessary in some circumstances that XSLT document will be processed as XML document. (Harold E.R., 1999)

**Table 7.60 Usage of document function without parameters**

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
  <params>
    <bgcolor>red</bgcolor><border>1</border>
  </params>
  <xsl:template match="/">
    <xsl:variable name="bgColor" select="document('')/*/params/bgcolor"/>
    <xsl:variable name="border" select="document('')/*/params/border"/>
    <TABLE border="{$border}" bgcolor="{$bgcolor}">
     document() usage</TABLE>
  </xsl:template>
</xsl:stylesheet>
```

In table 7.60, XPath document() function is used without parameters and accessed to XSL document which keeps information like in XML files. To traverse *bgcolor* and *border* elements which are defined in XSL document, *document('')/*/params/* X Path expression is used.

# CHAPTER EIGHT
# CONCLUSION

The Extensible Markup Language is a recommendation by the World Wide Web Consortium (W3C) for representing structured information in a text-based document. Although information without structured format (like in HTML) can be rendered via browsers, it can not be shared and processed globally for business purposes, because there exists no extra metadata (data about data) to describe what the information is. XML is a meta-language; it keeps both information and metadata. So, you can understand the semantic of information in XML documents by reading them simply.

XML's simplicity comes from W3C's compact XML specification framework. Shortly, from compact EBNF grammar. It is easy to compare an XML document to a BNF grammar. BNF grammars are not open to comments. An XML document fits rules or not, there are no other alternatives. XML 1.0 version has 89 EBNF rules.

Today, if there is need to "transfer data" from one platform to the other, XML technology is firstly thought as a solution. XML technology's readability, simplicity, extensibility, and lower implementation cost provides it to be stronger than traditional Electronic Data Interchange (EDI). This lower cost comes from XML's use of the Internet for data exchange, which is not easily achieved with EDI because of its framework. Business transactions over the Internet require interoperability while exchanging messages, and integrating applications. XML can be considered as a bridge which allows different systems to work together. It provides to standardize the business processes and transaction messages and also the method by which these messages are transmitted.

In addition to the data transfer, “application integration” is a most primarily task in programming; desktop applications integrate existing business applications, web applications integrate multiple Web Services and e-commerce sites integrate the other e-commerce sites or bussines systems like inventory and pricing. All of these applications can be designed by using the XML documents. Since XML is a simple text format and can be used many security and communication protocols, it is an ideal and effective choice for application integration.

“Data integration” is an important part of whole application integration. Data integration is simple, if there is only one database. But, everything is getting harder when more than one database like Oracle and SQL Server or mix of them is used.

If all the data for a given concept resides in a single data store, life is still simple. But if the data for a concept is collected from various media then you need to some integration to perform. XML and its related technologies are also designed to solve data integration problem and data integration becomes a lot easier. Technologies like XPath and XSLT can be used to splice, insert, or otherwise manipulate data from multiple sources by accessing all XML units like elements, attributes, comments and processing instructions.

XML is an effective, portable, easily customized data format that can easily sent over virtually any protocol. While it is just a data format, it can be used for many different purposes, ranging from messaging to RPC. The problem is that XML is being applied in every possible scenario even when it is not appropriate. This is primarily a problem in human nature in that people like to use new technologies for all problems.

Today XML is a powerful technology in B2B, web and integration processes. In the future, if web and communication becomes an important part of our life, you will not need to look far away for XML, it will stand just near you.

# REFERENCES

Gardner C. & Rendon Z., (2002), XSLT And XPATH, Prentice Hall Ptr.

Goldfarb C.F., (2001), XML Handbook, Pearson Education Inc

Harold E.R., (1999), XML Bible, IDG Books Worldwide Inc.

Pitts N., (2004), XML Black Book (2nd Edition), Coriolis Group LLC

WEB_1. (2004). Markup History. http://www.payer.de/xml/xml01.htm, 18/10/2003

WEB_2. (2003). GCA (Graphic Communications Association). http://www.gca.org, 18/10/2003

WEB_3. (2003). SGML. http://www.oasis-open.org/cover/sqprimerIntro.html, 29/10/2003

WEB_4. (2004). HTML Introduction. http://www.w3.org/TR/html4/, 02/11/2003

WEB_5. (2004). World Wide Web Consortium. http://www.w3c.org, 02/11/2003

WEB_6. (1999). General HTML Information. http://www.w3.org/MarkUp/html-spec/html-spec_3.html, 02/11/2003

WEB_7. (2000). Basic HTML Tags. http://sunsite.berkeley.edu/Web/basictags.html, 10/12/2003.

WEB_8. (2004). MathML. http://www.w3.org/TR/MathML2/, 20/12/2003.

WEB_9. (2004). CML. http://www.xml-cml.org/, 20/12/2003.

WEB_10. (2004).XML BNF. http://www.xml.com/pub/a/98/10/guide5.html 17/02/2004.

WEB_11. (2004). W3C XML 1.0 Recommendation. http://www.w3.org/TR/REC-xml/, 13/01/2004.

WEB_12. (2004). W3C XML 1.0. http://www.w3.org/TR/2004/REC-xml-20040204/, 13/01/2004.

WEB_13. (2004). The pros and cons of XML. http://www.zapthink.com/report.html?id=ZT-XMLPROCON, 20/02/2004.

WEB_14. (2004). DOM Intro. http://www.w3schools.com/dom/default.asp, 27/02/2004.

WEB_15. (2004). DOM Spec. http://www.w3.org/DOM/, 27/02/2004.

WEB_16. (2004). W3C XSLT 2.0. http://www.w3.org/TR/2003/WD-xslt20-20031112/, 05/04/2004.

WEB_17. (2004). W3C XPath 1.0. http://www.w3.org/TR/1999/REC-xpath-19991116, 23/04/2004.

WEB_18. (2004). SOAP. http://www.w3.org/TR/soap/, 05/01/2004.

WEB_19. (2004). XAML. http://www.XAML.net, 05/01/2004.

WEB_20. (2004). XPath. http://www.w3.org/TR/1999/REC-xpath-19991116, 14/05/2004.

WEB_21. (2004). XMLHTTP. http://www.w3.org/TR/1999/REC-xpath-19991116, 28/12/2003.

WEB_22. (2004). MSXML. http://www.perfectxml.com/MSXML.asp, 05/01/2004.

WEB_23. (2004). MSXML SDK. http://msdn.microsoft.com/library/ default.asp?url=/ library/en-us/ xmlsdk30/htm/xmmscxmloverview.asp, 28/12/2003.

WEB_24. (2004). EDI. http://www.tradanet.intnet.mu/EDIintro.htm, 30/12/2004.

WEB_25. (2004). BizTalk. http://www.microsoft.com/BizTalk, 30/12/2004.

WEB_26. (2004).UIML. http://www.uiml.org/, 05/01/2004.

WEB_27. (2004). XAML Control Sample. http://msdn.microsoft.com/library/ default.asp?url=/library/en-us/dnintlong/html/longhornch03.asp , 05/01/2004.

WEB_28. (2004). Microsoft Office Web Components. http://msdn.microsoft.com/ library/default.asp?url=/library/en-us/dno2kta/html/ofintrowbcom.asp, 09/01/2004.

WEB_29. (2004). W3C RDF Concepts And Syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, 02/05/2004.

WEB_30. (2004). XML Spy. http://www.altova.com/, 07/11/2003.

WEB_31. (2004). Macromedia Dreamweaver. http://www.macromedia.com/software/dreamweaver/, 07/11/2003.

WEB_32. (2004). CSS (Cascade Style Sheet). http://www.w3.org/Style/CSS/, 02/02/2004.

WEB_33. (2004). BNF (Backus Naur Form). http://www.scifac.ru.ac.za/compilers/cha05i.htm, 13/03/2004.

WEB_34. (2004). DTD Introduction. http://www.w3schools.com/dtd/, 03/04/2004.

WEB_35. (2004). CDATA Sections. http://www.w3schools.com/xml/xml_cdata.asp, 05/04/2004.

WEB_36. (2004). DTD in Detail. http://www.samspublishing.com/articles/article.asp?p=169516, 03/04/2004.

WEB_37. (2004). FPI (Formal Public Identifier). http://www.oasis-open.org/cover/tauber-fpi.html, 05/04/2004.

WEB_38. (2004). DTD Specification. http://xml.coverpages.org/xml-spec-report20.htm, 03/04/2004.

WEB_39. (2004). Entity Declaration. http://xmlwriter.net/xml_guide/entity_declaration, 14/04/2004.

WEB_40. (2004). PDF (Portable Document Format) . http://www.adobe.com/products/acrobat/adobepdf.html, 09/01/2004.

WEB_41. (2004). WAP (Wireless Application Protocol). http://www.wapforum.org, 11/02/2004.

WEB_42. (2004). WML (Wireless Markup Language). http://www.oasis-open.org/cover/wap-wml.html, 11/02/2004.