



DMA 1.0 Specification

Content Summary

1 Introduction 1-1

- 1.1 How to Use This Document 1-1
- 1.2 Objectives 1-2
- 1.3 History 1-3
- 1.4 Future Activities 1-3
- 1.5 Acknowledgments 1-4

2 Overview 2-1

- 2.1 The DMA Vision 2-1
- 2.2 The DMA Approach 2-4
- 2.3 DMA Concepts 2-4
- 2.4 Unification of Heterogeneous Document Management Systems 2-6

3 DMA Architecture 3-1

- 3.1 DMA Object Model 3-2
- 3.2 DMA Interface and Process Model 3-24
- 3.3 DMA Integration Model 3-31
- 3.4 DMA Distribution Model 3-52
- 3.5 DMA Query Model 3-55
- 3.6 DMA Containment Model 3-91
- 3.7 DMA Content Model 3-104
- 3.8 DMA Versioning Model 3-117
- 3.9 DMA Internationalization and Localization 3-126

4 DMA Reference 4-1

- 4.1 Objects and Properties 4-2
- 4.2 DMA Interfaces and Methods Reference 4-188
- 4.3 Macros 4-331
- 4.4 DMA Return Codes 4-334
- 4.5 Join Operators 4-339
- 4.6 DMA Well Known Query Operators 4-342
- 4.7 DMA Property DataTypes 4-353
- 4.8 Conformance 4-354

5 DMA Appendix 5-1

5.1 System & Document Space Connection Example 5-2

5.2 Content Examples 5-7

5.3 Pretty Print Example 5-18

5.4 Query Examples 5-24

5.5 Containment Examples 5-55

5.6 Versioning Example 5-65

5.7 DMA Glossary 5-69

Content

1 Introduction 1-1

1.1 How to Use This Document 1-1

1.1.1 Diagramming 1-2

1.2 Objectives 1-2

1.3 History 1-3

1.4 Future Activities 1-3

1.5 Acknowledgments 1-4

2 Overview 2-1

2.1 The DMA Vision 2-1

2.1.1 Uniform Access to Documents Anywhere 2-2

2.1.2 Self-describing Systems and Documents 2-2

2.1.3 Scalable Solutions 2-3

2.1.4 Expanded Collaboration Opportunities 2-3

2.1.5 Higher-Level Integration of Services and Applications 2-3

2.2 The DMA Approach 2-4

2.3 DMA Concepts 2-4

2.3.1 DMA Document Spaces 2-6

2.3.2 DMA Document Version Objects 2-6

2.4 Unification of Heterogeneous Document Management Systems 2-6

2.4.1 DMA Application Architecture 2-6

2.4.2 Provision for Uniformity and Unification 2-8

2.4.3 DMA Allows Evolution 2-8

2.4.4 Evolution of DMA capabilities and DMA-Compliant Components 2-9

3 DMA Architecture 3-1

3.1 DMA Object Model 3-2

3.1.1 Introduction 3-2

3.1.2 DMA Classes 3-3

3.1.2.1 Class Naming 3-3

3.1.2.2 Class Inheritance 3-3

3.1.2.3 Class Metadata 3-5

3.1.3 DMA Properties 3-7

3.1.3.1 Property Datatypes 3-7

3.1.3.1.1 Object-valued Properties 3-7

3.1.3.1.2 Multi-valued Properties 3-8

3.1.3.2 Property Naming 3-9

3.1.3.3 Property Metadata 3-10

3.1.4 Metadata Spaces	3-10
3.1.5 Persistence	3-12
3.1.5.1 Methods with Conflicting Intent	3-14
3.1.5.2 Locking Model	3-15
3.1.6 Objects and Implementations	3-18
3.1.7 Required and Optional Features	3-19
3.1.8 Extensibility	3-21
3.1.9 Evolution of Persistent Metadata	3-21
3.1.10 Document Space Connections	3-22
3.2 DMA Interface and Process Model	3-24
3.2.1 DMA Interface Model	3-24
3.2.1.1 COM Elements Used By DMA	3-24
3.2.1.1.1 GUID	3-24
3.2.1.1.2 Object and Interface Model	3-24
3.2.1.1.3 IUnknown Interface	3-26
3.2.1.1.4 Memory Management Rules	3-26
3.2.1.1.5 Error Handling	3-26
3.2.1.1.6 Error Result Codes	3-26
3.2.1.1.7 Transparent Remote Access	3-28
3.2.1.2 COM Elements Not Used By DMA	3-29
3.2.1.3 Additional Requirements on DMA Objects	3-29
3.2.1.3.1 Property Classes	3-29
3.2.1.3.2 Required interfaces	3-29
3.2.1.3.3 Object Generation	3-29
3.2.2 DMA Process Model	3-30
3.2.2.1 Objects Are Local to the Process	3-30
3.2.2.2 Thread Safety	3-30
3.2.2.3 Callbacks	3-30
3.3 DMA Integration Model	3-31
3.3.1 Introduction	3-31
3.3.1.1 Integration Model Principles	3-32
3.3.2 The Client View: Access to DMA Systems and Document Spaces	3-33
3.3.2.1 Application Connection to the System Manager	3-33
3.3.2.2 Accessing DMA Systems	3-34
3.3.2.3 Accessing DMA Document Spaces	3-35
3.3.2.4 Client Access Summary	3-37
3.3.2.5 Authentication	3-38
3.3.2.5.1 The DMA Authentication Protocol	3-39
3.3.2.5.2 DMA Authentication Schemes	3-39
3.3.2.5.2.1 DMA-BASIC Authentication	3-40
3.3.2.5.2.2 DMA-KERBEROS Authentication	3-40
3.3.2.5.2.3 Encryption	3-41
3.3.2.5.3 Automatic Authentication	3-41
3.3.2.1 Identifying and Accessing Persistable Document Space Objects	3-42
3.3.2.1.1 DMA OIID Syntax Specification	3-42
3.3.2.1.2 The DMA URL Scheme	3-43
3.3.2.1.2.1 DMA URL Examples:	3-44

3.3.2.1.2.2	<i>ODMA Encapsulation</i>	3-45
3.3.2.1.2.3	<i>BNF for DMA URLs</i>	3-45
3.3.3	DMA Service Objects	3-47
3.3.3.1	Service Object Registration	3-47
3.3.3.2	Service Object Activation	3-48
3.3.3.3	The System Object View: Integrating Systems for Access	3-49
3.3.3.3.1	Installing a System Object at the point of access	3-49
3.3.3.3.2	Typical Connection Entry into the System Service Object Implementation Program	3-49
3.3.3.4	The DocSpace Object View: Integrating DocSpace Objects into Systems	3-49
3.3.3.4.1	Installing a DocSpace Service Object at a Point of Service	3-49
3.3.3.4.2	Typical Connection Entry into the DocSpace Service Object Implementation Program	3-49
3.3.3.5	The Text Ordering Object View: Integrating Text Ordering Objects for Access	3-50
3.3.3.5.1	Installing a Text Ordering Service Object at a Point of Service	3-50
3.3.3.5.2	Typical Connection Entry into the Text Ordering Service Object Implementation Program	3-50
3.3.3.6	The Scope Factory Object View: Integrating a Scope Factory Object for Access	3-50
3.3.3.6.1	Installing a Scope Factory Service Object at a Point of Service	3-50
3.3.3.6.2	Typical Connection Entry into the Scope Factory Service Object Implementation Program	3-50
3.3.3.7	The OIID Parser Object View: Integrating OIID Parser Objects for Access	3-50
3.3.3.7.1	Installing a OIID Parser Service Object at a Point of Service	3-50
3.3.3.7.2	Typical Connection Entry into the OIID Parser Service Object Implementation Program	3-51
3.4	DMA Distribution Model	3-52
3.4.1	Client-Server Operation	3-52
3.4.2	Basic DMA Distribution	3-53
3.5	DMA Query Model	3-55
3.5.1	Introduction	3-55
3.5.2	Scope Object	3-56
3.5.3	Basic Search Formulation Process	3-56
3.5.4	The ExecuteSearch Method	3-57
3.5.5	Query Result Sets	3-58
3.5.6	Scopes and Their Creation	3-59
3.5.6.1	Properties of Scopes	3-59
3.5.7	Query Objects	3-61
3.5.7.1	Query Root Object	3-62
3.5.7.1.1	Subqueries	3-63
3.5.7.1.2	From Expression	3-64
3.5.7.1.3	Selections	3-67
3.5.7.1.4	Query Expression	3-67
3.5.7.1.4.1	<i>Query Execution With Merged Scopes</i>	3-67
3.5.7.1.4.2	<i>Partially Defined Query Expressions</i>	3-68
3.5.7.1.4.3	<i>Operations Across Document Spaces</i>	3-70
3.5.7.1.5	Orderings	3-70
3.5.7.1.6	Distinct Rows Requested	3-70
3.5.7.2	Example Query Object	3-71
3.5.8	Query Result Sets	3-72

3.5.8.1	Query Result Set Objects	3-72
3.5.8.2	Query Result Row Objects	3-73
3.5.8.2.1	Result Row Class Descriptions	3-75
3.5.8.2.1.1	<i>Duplicate Id's</i>	3-75
3.5.8.2.1.2	<i>Synthesized Object Properties</i>	3-75
3.5.9	Searching Across Multiple Repositories	3-77
3.5.9.1	Creating a Merged Scope Object	3-78
3.5.9.2	Merged Scopes Are an Optional Feature	3-79
3.5.9.3	Query Execution With Merged Scopes	3-79
3.5.9.4	Merging Metadata	3-81
3.5.9.4.1	Id's	3-82
3.5.9.4.2	Merging Searchable Classes	3-82
3.5.9.4.2.1	<i>Searchable Class Hierarchy</i>	3-83
3.5.9.4.3	Merge Options	3-84
3.5.9.4.4	Merging Class Descriptions	3-85
3.5.9.4.4.1	<i>Merging the Name Property Index Property</i>	3-85
3.5.9.4.4.2	<i>Merging Property Descriptions</i>	3-85
3.5.9.4.5	Merging Scope Properties	3-87
3.5.9.4.5.1	<i>Merging the Arbitrary Order By Property</i>	3-87
3.5.9.4.5.2	<i>Merging the Distinct Property</i>	3-87
3.5.9.4.5.3	<i>Merging the Collation Sequence Ids Property</i>	3-87
3.5.9.4.5.4	<i>Merging the Operators Property</i>	3-88
3.5.9.4.5.5	<i>Merging Query Construction Classes</i>	3-88
3.5.9.4.6	Merging Query Operand Descriptions	3-88
3.5.9.4.6.1	<i>Merging the Operand Data Type Property</i>	3-88
3.5.9.4.6.2	<i>Merging the Boolean Properties</i>	3-88
3.5.9.4.7	Merging Query Operator Descriptions	3-89
3.5.9.4.7.1	<i>Merging the Result Type Property</i>	3-89
3.5.9.4.7.2	<i>Merging the Is List Property</i>	3-89
3.5.9.4.7.3	<i>Merging the Is Safe To Eliminate Property</i>	3-89
3.5.9.4.7.4	<i>Merging the Join Participation Property</i>	3-89
3.5.9.4.7.5	<i>Miscellaneous Merging Rules</i>	3-90
3.6	DMA Containment Model	3-91
3.6.1	Introduction	3-91
3.6.2	DMA Containment Model Overview	3-92
3.6.3	Containment and the DMA Object Model	3-94
3.6.3.1	Containment Relationship Objects	3-96
3.6.4	Containment and Versioning	3-98
3.6.5	Containment and Persistence	3-98
3.6.5.1	Creating and Saving a Container	3-98
3.6.5.2	Modifying a Container	3-99
3.6.5.3	Deleting a Container	3-99
3.6.5.4	Maintaining Referential Integrity	3-100
3.6.6	Containment and Query	3-101
3.6.6.1	Navigational Queries	3-101
3.6.6.2	Associative Queries	3-102
3.7	DMA Content Model	3-104

3.7.1	Introduction	3-104
3.7.1.1	DocVersion	3-104
3.7.1.2	Rendition	3-105
3.7.1.3	Content Element	3-105
3.7.2	Modeling Document Content in DMA	3-105
3.7.3	Content Creation	3-108
3.7.3.1	DocVersion Creation	3-108
3.7.3.2	Rendition Creation	3-108
3.7.3.3	Content Element Creation	3-109
3.7.3.3.1	Determining the form of content capture	3-109
3.7.3.3.2	Creating ContentTransfer Objects	3-110
3.7.3.3.3	Creating ContentReference Objects	3-110
3.7.3.3.4	Making Content Persistent	3-110
3.7.4	Content Access and Modification	3-111
3.7.4.1	Locating and Accessing DocVersions	3-111
3.7.4.2	Accessing Renditions and Content Elements	3-111
3.7.4.3	Accessing Content Data	3-111
3.7.4.4	Modifying Content	3-112
3.7.4.5	Deleting Content	3-112
3.7.5	Content and the DMA Object Model	3-113
3.7.5.1	DocVersion	3-113
3.7.5.2	Rendition	3-114
3.7.5.3	Content Element	3-115
3.7.5.4	Content Transfer	3-115
3.7.5.4.1	IdmaContentTransfer	3-115
3.7.5.4.2	IdmaStream Interface	3-115
3.7.5.5	Content Reference	3-115
3.8	DMA Versioning Model	3-117
3.8.1	Objects Used for Versioning	3-118
3.8.2	Types of Versioning	3-120
3.8.2.1	Linear Versioning	3-120
3.8.2.2	Branch Versioning and Complex Configurations	3-122
3.8.2.3	Threaded Versioning	3-123
3.8.3	Basic Characteristics of DMA Versioning	3-124
3.8.4	DMA Versioning Operations	3-125
3.9	DMA Internationalization and Localization	3-126
3.9.1	Internationalization affected DMA Data Types	3-126
3.9.1.1	DmaString Data Type	3-126
3.9.1.2	DateTime Properties	3-127
3.9.2	Character Set Encodings	3-127
3.9.2.1	DMA Character Set Encoding Identifiers	3-128
3.9.3	Language Support	3-129
3.9.4	System Manager Internationalization	3-130
3.9.5	Text Collation	3-131
3.9.5.1	Well-known DMA Text Collation Sequences	3-131

4 DMA Reference 4-1

4.1 Objects and Properties 4-2

4.1.1 DMA Object Reference Column Descriptions 4-2

4.1.2 COM Objects Reference 4-4

4.1.2.2 SystemManager 4-4

4.1.2.2.1 Interfaces 4-4

4.1.2.2.2 Properties 4-4

4.1.2.2.3 Description 4-4

4.1.2.2.4 Property Descriptions 4-4

4.1.3 DMA Objects Reference 4-5

4.1.4 dmaClass_ClassDescription 4-8

4.1.4.1 Interfaces 4-8

4.1.4.2 Properties 4-8

4.1.4.3 Description 4-9

4.1.4.4 Property Descriptions 4-9

4.1.5 dmaClass_ConfigurationHistory 4-12

4.1.5.1 Interfaces 4-12

4.1.5.2 Properties 4-12

4.1.5.3 Property Descriptions 4-13

4.1.6 dmaClass_Containable 4-15

4.1.6.1 Interfaces 4-15

4.1.6.2 Properties 4-15

4.1.6.3 Property Descriptions 4-15

4.1.7 dmaClass_Container 4-18

4.1.7.1 Interfaces 4-18

4.1.7.2 Properties 4-18

4.1.7.3 Property Descriptions 4-19

4.1.8 dmaClass_ContainmentRelationship 4-21

4.1.8.1 Interfaces 4-21

4.1.8.2 Properties 4-21

4.1.8.3 Description 4-21

4.1.8.4 Property Descriptions 4-22

4.1.9 dmaClass_ContentElement 4-23

4.1.9.1 Interfaces 4-23

4.1.9.2 Properties 4-23

4.1.9.3 Description 4-23

4.1.9.4 Property Descriptions 4-23

4.1.10 dmaClass_ContentReference 4-25

4.1.10.1 Interfaces 4-25

4.1.10.2 Properties 4-25

4.1.10.3 Property Descriptions 4-25

4.1.11 dmaClass_ContentTransfer 4-27

4.1.11.1 Interfaces	4-27
4.1.11.2 Properties	4-27
4.1.11.3 Property Descriptions	4-27
4.1.12 dmaClass_DirectContainmentRelationship	4-29
4.1.12.1 Interfaces	4-29
4.1.12.2 Properties	4-29
4.1.12.3 Description	4-29
4.1.12.4 Property Descriptions	4-30
4.1.13 dmaClass_DMA	4-31
4.1.13.1 Interfaces	4-31
4.1.13.2 Properties	4-31
4.1.13.3 Description	4-31
4.1.13.4 Property Descriptions	4-31
4.1.14 dmaClass_DocSpace	4-34
4.1.14.1 Interfaces	4-34
4.1.14.2 Properties	4-34
4.1.14.3 Description	4-35
4.1.14.4 Property Descriptions	4-35
4.1.15 dmaClass_DocVersion	4-37
4.1.15.1 Interfaces	4-37
4.1.15.2 Properties	4-37
4.1.15.3 Description	4-38
4.1.15.4 Property Descriptions	4-38
4.1.16 dmaClass_Enumeration	4-40
4.1.16.1 Interfaces	4-40
4.1.16.2 Properties	4-40
4.1.16.3 Description	4-40
4.1.16.4 Property Descriptions	4-40
4.1.17 dmaClass_EnumerationOfObject	4-42
4.1.17.1 Interfaces	4-42
4.1.17.2 Properties	4-42
4.1.17.3 Description	4-42
4.1.17.4 Property Descriptions	4-42
4.1.18 dmaClass_List	4-44
4.1.18.1 Interfaces	4-44
4.1.18.2 Properties	4-44
4.1.18.3 Description	4-44
4.1.18.4 Property Descriptions	4-45
4.1.19 dmaClass_ListOfBinary	4-47
4.1.19.1 Interfaces	4-47
4.1.19.2 Properties	4-47
4.1.19.3 Description	4-48
4.1.19.4 Property Descriptions	4-48
4.1.20 dmaClass_ListOfBoolean	4-49

4.1.20.1 Interfaces	4-49
4.1.20.2 Properties	4-49
4.1.20.3 Description	4-50
4.1.20.4 Property Descriptions	4-50
4.1.21 dmaClass_ListOfDateTime	4-51
4.1.21.1 Interfaces	4-51
4.1.21.2 Properties	4-51
4.1.21.3 Description	4-52
4.1.21.4 Property Descriptions	4-52
4.1.22 dmaClass_ListOfFloat64	4-53
4.1.22.1 Interfaces	4-53
4.1.22.2 Properties	4-53
4.1.22.3 Description	4-54
4.1.22.4 Property Descriptions	4-54
4.1.23 dmaClass_ListOfId	4-55
4.1.23.1 Interfaces	4-55
4.1.23.2 Properties	4-55
4.1.23.3 Description	4-56
4.1.23.4 Property Descriptions	4-56
4.1.24 dmaClass_ListOfInteger32	4-57
4.1.24.1 Interfaces	4-57
4.1.24.2 Properties	4-57
4.1.24.3 Description	4-58
4.1.24.4 Property Descriptions	4-58
4.1.25 dmaClass_ListOfObject	4-59
4.1.25.1 Interfaces	4-59
4.1.25.2 Properties	4-59
4.1.25.3 Description	4-60
4.1.25.4 Property Descriptions	4-60
4.1.26 dmaClass_ListOfString	4-62
4.1.26.1 Interfaces	4-62
4.1.26.2 Properties	4-62
4.1.26.3 Description	4-63
4.1.26.4 Property Descriptions	4-63
4.1.27 dmaClass_Metadata	4-64
4.1.27.1 Interfaces	4-64
4.1.27.2 Properties	4-64
4.1.27.3 Property Descriptions	4-64
4.1.28 dmaClass_PropertyDescription	4-66
4.1.28.1 Interfaces	4-66
4.1.28.2 Properties	4-66
4.1.28.3 Description	4-67
4.1.28.4 Property Descriptions	4-67
4.1.29 dmaClass_PropertyDescriptionBinary	4-70

4.1.29.1 Interfaces	4-70
4.1.29.2 Properties	4-70
4.1.29.3 Description	4-71
4.1.29.4 Property Descriptions	4-71
4.1.30 dmaClass_PropertyDescriptionBoolean	4-73
4.1.30.1 Interfaces	4-73
4.1.30.2 Properties	4-73
4.1.30.3 Description	4-74
4.1.30.4 Property Descriptions	4-74
4.1.31 dmaClass_PropertyDescriptionDateTime	4-76
4.1.31.1 Interfaces	4-76
4.1.31.2 Properties	4-76
4.1.31.3 Description	4-77
4.1.31.4 Property Descriptions	4-77
4.1.32 dmaClass_PropertyDescriptionFloat64	4-80
4.1.32.1 Interfaces	4-80
4.1.32.2 Properties	4-80
4.1.32.3 Description	4-81
4.1.32.4 Property Descriptions	4-81
4.1.33 dmaClass_PropertyDescriptionId	4-84
4.1.33.1 Interfaces	4-84
4.1.33.2 Properties	4-84
4.1.33.3 Description	4-85
4.1.33.4 Property Descriptions	4-85
4.1.34 dmaClass_PropertyDescriptionInteger32	4-87
4.1.34.1 Interfaces	4-87
4.1.34.2 Properties	4-87
4.1.34.3 Description	4-88
4.1.34.4 Property Descriptions	4-88
4.1.35 dmaClass_PropertyDescriptionObject	4-91
4.1.35.1 Interfaces	4-91
4.1.35.2 Properties	4-91
4.1.35.3 Description	4-92
4.1.35.4 Property Descriptions	4-92
4.1.36 dmaClass_PropertyDescriptionString	4-95
4.1.36.1 Interfaces	4-95
4.1.36.2 Properties	4-95
4.1.36.3 Description	4-96
4.1.36.4 Property Descriptions	4-96
4.1.37 dmaClass_Query	4-99
4.1.37.1 Interfaces	4-99
4.1.37.2 Properties	4-99
4.1.37.3 Description	4-99
4.1.37.4 Property Descriptions	4-100

4.1.38 dmaClass_QueryConstant	4-102
4.1.38.1 Interfaces	4-102
4.1.38.2 Properties	4-102
4.1.38.3 Property Descriptions	4-102
4.1.39 dmaClass_QueryConstantBinaries	4-104
4.1.39.1 Interfaces	4-104
4.1.39.2 Properties	4-104
4.1.39.3 Property Descriptions	4-104
4.1.40 dmaClass_QueryConstantBinary	4-106
4.1.40.1 Interfaces	4-106
4.1.40.2 Properties	4-106
4.1.40.3 Property Descriptions	4-106
4.1.41 dmaClass_QueryConstantBoolean	4-108
4.1.41.1 Interfaces	4-108
4.1.41.2 Properties	4-108
4.1.41.3 Property Descriptions	4-108
4.1.42 dmaClass_QueryConstantBooleans	4-110
4.1.42.1 Interfaces	4-110
4.1.42.2 Properties	4-110
4.1.42.3 Property Descriptions	4-110
4.1.43 dmaClass_QueryConstantDateTime	4-112
4.1.43.1 Interfaces	4-112
4.1.43.2 Properties	4-112
4.1.43.3 Property Descriptions	4-112
4.1.44 dmaClass_QueryConstantDateTimes	4-114
4.1.44.1 Interfaces	4-114
4.1.44.2 Properties	4-114
4.1.44.3 Property Descriptions	4-114
4.1.45 dmaClass_QueryConstantFloat64	4-116
4.1.45.1 Interfaces	4-116
4.1.45.2 Properties	4-116
4.1.45.3 Property Descriptions	4-116
4.1.46 dmaClass_QueryConstantFloat64s	4-118
4.1.46.1 Interfaces	4-118
4.1.46.2 Properties	4-118
4.1.46.3 Property Descriptions	4-118
4.1.47 dmaClass_QueryConstantId	4-120
4.1.47.1 Interfaces	4-120
4.1.47.2 Properties	4-120
4.1.47.3 Property Descriptions	4-120
4.1.48 dmaClass_QueryConstantIds	4-122
4.1.48.1 Interfaces	4-122
4.1.48.2 Properties	4-122
4.1.48.3 Property Descriptions	4-122

4.1.49 dmaClass_QueryConstantInteger32	4-124
4.1.49.1 Interfaces	4-124
4.1.49.2 Properties	4-124
4.1.49.3 Property Descriptions	4-124
4.1.50 dmaClass_QueryConstantInteger32s	4-126
4.1.50.1 Interfaces	4-126
4.1.50.2 Properties	4-126
4.1.50.3 Property Descriptions	4-126
4.1.51 dmaClass_QueryConstantString	4-128
4.1.51.1 Interfaces	4-128
4.1.51.2 Properties	4-128
4.1.51.3 Property Descriptions	4-128
4.1.52 dmaClass_QueryConstantStrings	4-130
4.1.52.1 Interfaces	4-130
4.1.52.2 Properties	4-130
4.1.52.3 Property Descriptions	4-130
4.1.53 dmaClass_QueryJoinOperator	4-132
4.1.53.1 Interfaces	4-132
4.1.53.2 Properties	4-132
4.1.53.3 Description	4-132
4.1.53.4 Property Descriptions	4-133
4.1.54 dmaClass_QueryNode	4-134
4.1.54.1 Interfaces	4-134
4.1.54.2 Properties	4-134
4.1.54.3 Description	4-134
4.1.54.4 Property Descriptions	4-134
4.1.55 dmaClass_QueryNonterminalNode	4-136
4.1.55.1 Interfaces	4-136
4.1.55.2 Properties	4-136
4.1.55.3 Property Descriptions	4-136
4.1.56 dmaClass_QueryOperandDescription	4-138
4.1.56.1 Interfaces	4-138
4.1.56.2 Properties	4-138
4.1.56.3 Description	4-138
4.1.56.4 Property Descriptions	4-139
4.1.57 dmaClass_QueryOperator	4-141
4.1.57.1 Interfaces	4-141
4.1.57.2 Properties	4-141
4.1.57.3 Property Descriptions	4-141
4.1.58 dmaClass_QueryOperatorDescription	4-143
4.1.58.1 Interfaces	4-143
4.1.58.2 Properties	4-143
4.1.58.3 Description	4-144
4.1.58.4 Property Descriptions	4-144

4.1.59 dmaClass_QueryOrderByNode	4-148
4.1.59.1 Interfaces	4-148
4.1.59.2 Properties	4-148
4.1.59.3 Property Descriptions	4-148
4.1.60 dmaClass_QueryProperty	4-150
4.1.60.1 Interfaces	4-150
4.1.60.2 Properties	4-150
4.1.60.3 Property Descriptions	4-150
4.1.61 dmaClass_QueryResultRow	4-152
4.1.61.1 Interfaces	4-152
4.1.61.2 Properties	4-152
4.1.61.3 Description	4-152
4.1.61.4 Property Descriptions	4-152
4.1.62 dmaClass_QueryResultSet	4-154
4.1.62.1 Interfaces	4-154
4.1.62.2 Properties	4-154
4.1.62.3 Property Descriptions	4-154
4.1.63 dmaClass_QueryRoot	4-156
4.1.63.1 Interfaces	4-156
4.1.63.2 Properties	4-156
4.1.63.3 Property Descriptions	4-156
4.1.64 dmaClass_QuerySearchableClass	4-159
4.1.64.1 Interfaces	4-159
4.1.64.2 Properties	4-159
4.1.64.3 Property Descriptions	4-159
4.1.65 dmaClass_ReferentialContainmentRelationship	4-162
4.1.65.1 Interfaces	4-162
4.1.65.2 Properties	4-162
4.1.65.3 Description	4-162
4.1.65.4 Property Descriptions	4-163
4.1.66 dmaClass_Relationship	4-164
4.1.66.1 Interfaces	4-164
4.1.66.2 Properties	4-164
4.1.66.3 Description	4-164
4.1.66.4 Property Descriptions	4-165
4.1.67 dmaClass_Rendition	4-166
4.1.67.1 Interfaces	4-166
4.1.67.2 Properties	4-166
4.1.67.3 Description	4-166
4.1.67.4 Property Descriptions	4-167
4.1.68 dmaClass_Reservation	4-169
4.1.68.1 Interfaces	4-169
4.1.68.2 Properties	4-169
4.1.68.3 Property Descriptions	4-169

4.1.69 dmaClass_Scope	4-171
4.1.69.1 Interfaces	4-171
4.1.69.2 Properties	4-171
4.1.69.3 Description	4-172
4.1.69.4 Property Descriptions	4-172
4.1.70 dmaClass_System	4-175
4.1.70.1 Interfaces	4-175
4.1.70.2 Properties	4-175
4.1.70.3 Description	4-176
4.1.70.4 Property Descriptions	4-176
4.1.71 dmaClass_Versionable	4-178
4.1.71.1 Interfaces	4-178
4.1.71.2 Properties	4-178
4.1.71.3 Description	4-179
4.1.71.4 Property Descriptions	4-179
4.1.72 dmaClass_VersionDescription	4-181
4.1.72.1 Interfaces	4-181
4.1.72.2 Properties	4-181
4.1.72.3 Description	4-182
4.1.72.4 Property Descriptions	4-182
4.1.73 dmaClass_VersionSeries	4-184
4.1.73.1 Interfaces	4-184
4.1.73.2 Properties	4-184
4.1.73.3 Description	4-185
4.1.73.4 Property Descriptions	4-185
4.2 DMA Interfaces and Methods Reference	4-188
4.2.1 Introduction and Conventions	4-188
4.2.2 Free Standing Function and Entry-Points	4-189
4.2.2.1 DMA_GetServiceObject	4-189
4.2.2.1.1 Syntax	4-189
4.2.2.1.2 Parameters	4-189
4.2.2.1.3 Description	4-192
4.2.2.1.4 Return Values	4-192
4.2.2.2 dmaConnectSystemManager	4-193
4.2.2.2.1 Syntax	4-193
4.2.2.2.2 Parameters	4-193
4.2.2.2.3 Description	4-195
4.2.2.2.4 Return Values	4-196
4.2.3 IdmaAuthentication	4-196
4.2.3.1 IdmaAuthentication::AuthenticateUser	4-196
4.2.3.1.1 Syntax	4-196
4.2.3.1.2 Parameters	4-196
4.2.3.1.3 Description	4-197
4.2.3.1.4 DMA Authentication Schemes	4-197
4.2.3.1.5 DMA-BASIC Authentication	4-198
4.2.3.1.6 DMA-KERBEROS Authentication	4-198

4.2.3.1.7 Return Values	4-199
4.2.3.2 IdmaAuthentication::AutoAuthenticateUser	4-199
4.2.3.2.1 Syntax	4-199
4.2.3.2.2 Description	4-199
4.2.3.2.3 Return Values	4-199
4.2.4 IdmaBatch	4-200
4.2.4.1 IdmaBatch::ExecuteChanges	4-200
4.2.4.1.1 Syntax	4-200
4.2.4.1.2 Parameters	4-200
4.2.4.1.3 Description	4-200
4.2.4.1.4 Return Values	4-201
4.2.5 IdmaClassDescription	4-202
4.2.5.1 IdmaClassDescription::CreateInstance	4-202
4.2.5.1.1 Syntax	4-202
4.2.5.1.2 Parameters	4-202
4.2.5.1.3 Description	4-203
4.2.5.1.4 Return Values	4-203
4.2.5.2 IdmaClassDescription::IsOfClass	4-203
4.2.5.2.1 Syntax	4-203
4.2.5.2.2 Parameters	4-203
4.2.5.2.3 Description	4-203
4.2.5.2.4 Return Values	4-204
4.2.6 IdmaConnection	4-204
4.2.6.1 IdmaConnection::ApplyLock	4-204
4.2.6.1.1 Syntax	4-204
4.2.6.1.2 Parameters	4-204
4.2.6.1.3 Description	4-204
4.2.6.1.4 Return Values	4-205
4.2.6.2 IdmaConnection::ExecuteChange	4-205
4.2.6.2.1 Syntax	4-205
4.2.6.2.2 Parameters	4-206
4.2.6.2.3 Description	4-206
4.2.6.2.4 Return Values	4-208
4.2.6.3 IdmaConnection::ExecuteRefresh	4-208
4.2.6.3.1 Syntax	4-208
4.2.6.3.2 Description	4-209
4.2.6.3.3 Return Values	4-209
4.2.6.4 IdmaConnection::IsCurrent	4-209
4.2.6.4.1 Syntax	4-209
4.2.6.4.2 Parameters	4-209
4.2.6.4.3 Description	4-209
4.2.6.4.4 Return Values	4-210
4.2.6.5 IdmaConnection::RemoveLock	4-210
4.2.6.5.1 Syntax	4-210
4.2.6.5.2 Description	4-210
4.2.6.5.3 Return Values	4-210
4.2.6.6 IdmaConnection::SetDeletePending	4-211
4.2.6.6.1 Syntax	4-211
4.2.6.6.2 Parameters	4-211

4.2.6.6.3 Description	4-211
4.2.6.6.4 Return Values	4-211
4.2.6.6.5 Deferred Return Values	4-211
4.2.7 IdmaContentTransfer	4-212
4.2.7.1 IdmaContentTransfer::CopyToResource	4-212
4.2.7.1.1 Syntax	4-212
4.2.7.1.2 Parameters	4-212
4.2.7.1.3 Description	4-212
4.2.7.2 IdmaContentTransfer::GetResourceName	4-213
4.2.7.2.1 Syntax	4-213
4.2.7.2.2 Parameters	4-213
4.2.7.2.3 Description	4-214
4.2.7.2.4 Return Values	4-214
4.2.7.3 IdmaContentTransfer::GetStream	4-214
4.2.7.3.1 Syntax	4-214
4.2.7.3.2 Parameters	4-215
4.2.7.3.3 Description	4-215
4.2.7.3.4 Return Values	4-215
4.2.7.4 IdmaContentTransfer::SetCaptureResource	4-215
4.2.7.4.1 Syntax	4-215
4.2.7.4.2 Parameters	4-216
4.2.7.4.3 Description	4-216
4.2.7.4.4 Return Values	4-216
4.2.7.4.5 Deferred Return Values	4-217
4.2.7.5 IdmaContentTransfer::SetCaptureStream	4-217
4.2.7.5.1 Syntax	4-217
4.2.7.5.2 Parameters	4-217
4.2.7.5.3 Description	4-217
4.2.7.5.4 Return Values	4-218
4.2.7.5.5 Deferred Return Values	4-218
4.2.8 IdmaDocSpace	4-218
4.2.8.1 IdmaDocSpace::ConnectAndLockObject	4-218
4.2.8.1.1 Syntax	4-218
4.2.8.1.2 Parameters	4-218
4.2.8.1.3 Description	4-219
4.2.8.1.4 Return Values	4-219
4.2.8.2 IdmaDocSpace::ConnectObject	4-219
4.2.8.2.1 Syntax	4-219
4.2.8.2.2 Parameters	4-220
4.2.8.2.3 Description	4-220
4.2.8.2.4 Return Values	4-220
4.2.8.3 IdmaDocSpace::ExecuteDisconnect	4-221
4.2.8.3.1 Syntax	4-221
4.2.8.3.2 Description	4-221
4.2.8.3.3 Return Values	4-221
4.2.8.4 IdmaDocSpace::GetScope	4-221
4.2.8.4.1 Syntax	4-222
4.2.8.4.2 Parameters	4-222
4.2.8.4.3 Description	4-222

4.2.8.4.4	Return Values	4-222
4.2.9	IdmaEditList	4-223
4.2.9.1	IdmaEditList::DeleteElement	4-223
4.2.9.1.1	Syntax	4-223
4.2.9.1.2	Parameters	4-223
4.2.9.1.3	Description	4-223
4.2.9.1.4	Return Values	4-223
4.2.9.1.5	Deferred Return Values	4-224
4.2.9.2	IdmaEditList::TruncateList	4-224
4.2.9.2.1	Syntax	4-224
4.2.9.2.2	Parameters	4-224
4.2.9.2.3	Description	4-224
4.2.9.2.4	Return Values	4-224
4.2.9.2.5	Deferred Return Values	4-224
4.2.10	IdmaEditListOfBinary	4-225
4.2.10.1	IdmaEditListOfBinary::InsertBinary	4-225
4.2.10.1.1	Syntax	4-225
4.2.10.1.2	Parameters	4-225
4.2.10.1.3	Description	4-225
4.2.10.1.4	Return Values	4-225
4.2.10.1.5	Deferred Return Values	4-226
4.2.10.2	IdmaEditListOfBinary::ReplaceBinary	4-226
4.2.10.2.1	Syntax	4-226
4.2.10.2.2	Parameters	4-226
4.2.10.2.3	Description	4-226
4.2.10.2.4	Return Values	4-226
4.2.10.2.5	Deferred Return Values	4-227
4.2.11	IdmaEditListOfBoolean	4-227
4.2.11.1	IdmaEditListOfBoolean::InsertBoolean	4-227
4.2.11.1.1	Syntax	4-227
4.2.11.1.2	Parameters	4-227
4.2.11.1.3	Description	4-227
4.2.11.1.4	Return Values	4-227
4.2.11.1.5	Deferred Return Values	4-228
4.2.11.2	IdmaEditListOfBoolean::ReplaceBoolean	4-228
4.2.11.2.1	Syntax	4-228
4.2.11.2.2	Parameters	4-228
4.2.11.2.3	Description	4-228
4.2.11.2.4	Return Values	4-228
4.2.11.2.5	Deferred Return Values	4-229
4.2.12	IdmaEditListOfDateTime	4-229
4.2.12.1	IdmaEditListOfDateTime::InsertDateTime	4-229
4.2.12.1.1	Syntax	4-229
4.2.12.1.2	Parameters	4-229
4.2.12.1.3	Description	4-229
4.2.12.1.4	Return Values	4-230
4.2.12.1.5	Deferred Return Values	4-230
4.2.12.2	IdmaEditListOfDateTime::ReplaceDateTime	4-230
4.2.12.2.1	Syntax	4-230

- 4.2.12.2.2 Parameters 4-230
 - 4.2.12.2.3 Description 4-230
 - 4.2.12.2.4 Return Values 4-231
 - 4.2.12.2.5 Deferred Return Values 4-231
- 4.2.13 IdmaEditListOfFloat64 4-231
 - 4.2.13.1 IdmaEditListOfFloat64::InsertFloat64 4-231
 - 4.2.13.1.1 Syntax 4-231
 - 4.2.13.1.2 Parameters 4-231
 - 4.2.13.1.3 Description 4-232
 - 4.2.13.1.4 Return Values 4-232
 - 4.2.13.1.5 Deferred Return Values 4-232
 - 4.2.13.2 IdmaEditListOfFloat64::ReplaceFloat64 4-232
 - 4.2.13.2.1 Syntax 4-232
 - 4.2.13.2.2 Parameters 4-232
 - 4.2.13.2.3 Description 4-233
 - 4.2.13.2.4 Return Values 4-233
 - 4.2.13.2.5 Deferred Return Values 4-233
- 4.2.14 IdmaEditListOfId 4-233
 - 4.2.14.1 IdmaEditListOfId::InsertId 4-233
 - 4.2.14.1.1 Syntax 4-233
 - 4.2.14.1.2 Parameters 4-234
 - 4.2.14.1.3 Description 4-234
 - 4.2.14.1.4 Return Values 4-234
 - 4.2.14.1.5 Deferred Return Values 4-234
 - 4.2.14.2 IdmaEditListOfId::ReplaceId 4-234
 - 4.2.14.2.1 Syntax 4-234
 - 4.2.14.2.2 Parameters 4-235
 - 4.2.14.2.3 Description 4-235
 - 4.2.14.2.4 Return Values 4-235
 - 4.2.14.2.5 Deferred Return Values 4-235
- 4.2.15 IdmaEditListOfInteger32 4-235
 - 4.2.15.1 IdmaEditListOfInteger32::InsertInteger32 4-235
 - 4.2.15.1.1 Syntax 4-236
 - 4.2.15.1.2 Parameters 4-236
 - 4.2.15.1.3 Description 4-236
 - 4.2.15.1.4 Return Values 4-236
 - 4.2.15.1.5 Deferred Return Values 4-236
 - 4.2.15.2 IdmaEditListOfInteger32::ReplaceInteger32 4-236
 - 4.2.15.2.1 Syntax 4-237
 - 4.2.15.2.2 Parameters 4-237
 - 4.2.15.2.3 Description 4-237
 - 4.2.15.2.4 Return Values 4-237
 - 4.2.15.2.5 Deferred Return Values 4-237
- 4.2.16 IdmaEditListOfObject 4-237
 - 4.2.16.1 IdmaEditListOfObject::InsertObject 4-238
 - 4.2.16.1.1 Syntax 4-238
 - 4.2.16.1.2 Parameters 4-238
 - 4.2.16.1.3 Description 4-238
 - 4.2.16.1.4 Return Values 4-238

4.2.16.1.5 Deferred Return Values	4-238
4.2.16.2 IdmaEditListOfObject::ReplaceObject	4-239
4.2.16.2.1 Syntax	4-239
4.2.16.2.2 Parameters	4-239
4.2.16.2.3 Description	4-239
4.2.16.2.4 Return Values	4-239
4.2.16.2.5 Deferred Return Values	4-239
4.2.17 IdmaEditListOfString	4-240
4.2.17.1 IdmaEditListOfString::InsertString	4-240
4.2.17.1.1 Syntax	4-240
4.2.17.1.2 Parameters	4-240
4.2.17.1.3 Description	4-240
4.2.17.1.4 Return Values	4-240
4.2.17.1.5 Deferred Return Values	4-241
4.2.17.2 IdmaEditListOfString::ReplaceString	4-241
4.2.17.2.1 Syntax	4-241
4.2.17.2.2 Parameters	4-241
4.2.17.2.3 Description	4-241
4.2.17.2.4 Return Values	4-241
4.2.17.2.5 Deferred Return Values	4-242
4.2.18 IdmaEditProperties	4-242
4.2.18.1 IdmaEditProperties::DeletePropValById	4-242
4.2.18.1.1 Syntax	4-242
4.2.18.1.2 Parameters	4-242
4.2.18.1.3 Description	4-242
4.2.18.1.4 Return Values	4-243
4.2.18.1.5 Deferred Return Values	4-243
4.2.18.2 IdmaEditProperties::DeletePropValByIndex	4-243
4.2.18.2.1 Syntax	4-243
4.2.18.2.2 Parameters	4-243
4.2.18.2.3 Description	4-243
4.2.18.2.4 Return Values	4-244
4.2.18.2.5 Deferred Return Values	4-244
4.2.18.3 IdmaEditProperties::PutPropValBinaryById	4-244
4.2.18.3.1 Syntax	4-244
4.2.18.3.2 Parameters	4-244
4.2.18.3.3 Description	4-244
4.2.18.3.4 Return Values	4-245
4.2.18.3.5 Deferred Return Values	4-245
4.2.18.4 IdmaEditProperties::PutPropValBinaryByIndex	4-245
4.2.18.4.1 Syntax	4-245
4.2.18.4.2 Parameters	4-245
4.2.18.4.3 Description	4-245
4.2.18.4.4 Return Values	4-246
4.2.18.4.5 Deferred Return Values	4-246
4.2.18.5 IdmaEditProperties::PutPropValBooleanById	4-246
4.2.18.5.1 Syntax	4-246
4.2.18.5.2 Parameters	4-246
4.2.18.5.3 Description	4-246
4.2.18.5.4 Return Values	4-247

4.2.18.5.5 Deferred Return Values	4-247
4.2.18.6 IdmaEditProperties::PutPropValBooleanByIndex	4-247
4.2.18.6.1 Syntax	4-247
4.2.18.6.2 Parameters	4-247
4.2.18.6.3 Description	4-247
4.2.18.6.4 Return Values	4-248
4.2.18.6.5 Deferred Return Values	4-248
4.2.18.7 IdmaEditProperties::PutPropValDateTimeByld	4-248
4.2.18.7.1 Syntax	4-248
4.2.18.7.2 Parameters	4-248
4.2.18.7.3 Description	4-248
4.2.18.7.4 Return Values	4-249
4.2.18.7.5 Deferred Return Values	4-249
4.2.18.8 IdmaEditProperties::PutPropValDateTimeByIndex	4-249
4.2.18.8.1 Syntax	4-249
4.2.18.8.2 Parameters	4-249
4.2.18.8.3 Description	4-249
4.2.18.8.4 Return Values	4-250
4.2.18.8.5 Deferred Return Values	4-250
4.2.18.9 IdmaEditProperties::PutPropValFloat64Byld	4-250
4.2.18.9.1 Syntax	4-250
4.2.18.9.2 Parameters	4-250
4.2.18.9.3 Description	4-250
4.2.18.9.4 Return Values	4-251
4.2.18.9.5 Deferred Return Values	4-251
4.2.18.10 IdmaEditProperties::PutPropValFloat64ByIndex	4-251
4.2.18.10.1 Syntax	4-251
4.2.18.10.2 Parameters	4-251
4.2.18.10.3 Description	4-251
4.2.18.10.4 Return Values	4-252
4.2.18.10.5 Deferred Return Values	4-252
4.2.18.11 IdmaEditProperties::PutPropValldByld	4-252
4.2.18.11.1 Syntax	4-252
4.2.18.11.2 Parameters	4-252
4.2.18.11.3 Description	4-252
4.2.18.11.4 Return Values	4-253
4.2.18.11.5 Deferred Return Values	4-253
4.2.18.12 IdmaEditProperties::PutPropValldByIndex	4-253
4.2.18.12.1 Syntax	4-253
4.2.18.12.2 Parameters	4-253
4.2.18.12.3 Description	4-253
4.2.18.12.4 Return Values	4-254
4.2.18.12.5 Deferred Return Values	4-254
4.2.18.13 IdmaEditProperties::PutPropValInteger32Byld	4-254
4.2.18.13.1 Syntax	4-254
4.2.18.13.2 Parameters	4-254
4.2.18.13.3 Description	4-254
4.2.18.13.4 Return Values	4-255
4.2.18.13.5 Deferred Return Values	4-255
4.2.18.14 IdmaEditProperties::PutPropValInteger32ByIndex	4-255

- 4.2.18.14.1 Syntax 4-255
 - 4.2.18.14.2 Parameters 4-255
 - 4.2.18.14.3 Description 4-255
 - 4.2.18.14.4 Return Values 4-256
 - 4.2.18.14.5 Deferred Return Values 4-256
- 4.2.18.15 IdmaEditProperties::PutPropValObjectById 4-256
 - 4.2.18.15.1 Syntax 4-256
 - 4.2.18.15.2 Parameters 4-256
 - 4.2.18.15.3 Description 4-256
 - 4.2.18.15.4 Return Values 4-257
 - 4.2.18.15.5 Deferred Return Values 4-257
- 4.2.18.16 IdmaEditProperties::PutPropValObjectByIndex 4-257
 - 4.2.18.16.1 Syntax 4-257
 - 4.2.18.16.2 Parameters 4-257
 - 4.2.18.16.3 Description 4-258
 - 4.2.18.16.4 Return Values 4-258
 - 4.2.18.16.5 Deferred Return Values 4-258
- 4.2.18.17 IdmaEditProperties::PutPropValStringById 4-258
 - 4.2.18.17.1 Syntax 4-258
 - 4.2.18.17.2 Parameters 4-259
 - 4.2.18.17.3 Description 4-259
 - 4.2.18.17.4 Return Values 4-259
 - 4.2.18.17.5 Deferred Return Values 4-259
- 4.2.18.18 IdmaEditProperties::PutPropValStringByIndex 4-259
 - 4.2.18.18.1 Syntax 4-259
 - 4.2.18.18.2 Parameters 4-260
 - 4.2.18.18.3 Description 4-260
 - 4.2.18.18.4 Return Values 4-260
 - 4.2.18.18.5 Deferred Return Values 4-260
- 4.2.19 IdmaEnumOfObject 4-260
 - 4.2.19.1 IdmaEnumOfObject::GetNextObject 4-260
 - 4.2.19.1.1 Syntax 4-261
 - 4.2.19.1.2 Parameters 4-261
 - 4.2.19.1.3 Description 4-261
 - 4.2.19.1.4 Return Values 4-262
- 4.2.20 IdmaList 4-262
 - 4.2.20.1 IdmaList::GetElementCount 4-262
 - 4.2.20.1.1 Syntax 4-262
 - 4.2.20.1.2 Parameters 4-262
 - 4.2.20.1.3 Description 4-263
 - 4.2.20.1.4 Return Values 4-263
- 4.2.21 IdmaListOfBinary 4-263
 - 4.2.21.1 IdmaListOfBinary::GetBinary 4-263
 - 4.2.21.1.1 Syntax 4-263
 - 4.2.21.1.2 Parameters 4-263
 - 4.2.21.1.3 Description 4-263
 - 4.2.21.1.4 Return Values 4-264
- 4.2.22 IdmaListOfBoolean 4-264
 - 4.2.22.1 IdmaListOfBoolean::GetBoolean 4-264

- 4.2.22.1.1 Syntax 4-264
 - 4.2.22.1.2 Parameters 4-264
 - 4.2.22.1.3 Description 4-264
 - 4.2.22.1.4 Return Values 4-265
- 4.2.23 IdmaListOfDateTime 4-265
 - 4.2.23.1 IdmaListOfDateTime::GetDateTime 4-265
 - 4.2.23.1.1 Syntax 4-265
 - 4.2.23.1.2 Parameters 4-265
 - 4.2.23.1.3 Description 4-265
 - 4.2.23.1.4 Return Values 4-265
- 4.2.24 IdmaListOfFloat64 4-266
 - 4.2.24.1 IdmaListOfFloat64::GetFloat64 4-266
 - 4.2.24.1.1 Syntax 4-266
 - 4.2.24.1.2 Parameters 4-266
 - 4.2.24.1.3 Description 4-266
 - 4.2.24.1.4 Return Values 4-266
- 4.2.25 IdmaListOfId 4-267
 - 4.2.25.1 IdmaListOfId::GetId 4-267
 - 4.2.25.1.1 Syntax 4-267
 - 4.2.25.1.2 Parameters 4-267
 - 4.2.25.1.3 Description 4-267
 - 4.2.25.1.4 Return Values 4-267
- 4.2.26 IdmaListOfInteger32 4-267
 - 4.2.26.1 IdmaListOfInteger32::GetInteger32 4-268
 - 4.2.26.1.1 Syntax 4-268
 - 4.2.26.1.2 Parameters 4-268
 - 4.2.26.1.3 Description 4-268
 - 4.2.26.1.4 Return Values 4-268
- 4.2.27 IdmaListOfObject 4-268
 - 4.2.27.1 IdmaListOfObject::GetObject 4-268
 - 4.2.27.1.1 Syntax 4-269
 - 4.2.27.1.2 Parameters 4-269
 - 4.2.27.1.3 Description 4-269
 - 4.2.27.1.4 Return Values 4-269
- 4.2.28 IdmaListOfString 4-270
 - 4.2.28.1 IdmaListOfString::GetString 4-270
 - 4.2.28.1.1 Syntax 4-270
 - 4.2.28.1.2 Parameters 4-270
 - 4.2.28.1.3 Description 4-270
 - 4.2.28.1.4 Return Values 4-270
- 4.2.29 IdmaObject 4-270
 - 4.2.29.1 IdmaObject::IsOfClass 4-270
 - 4.2.29.1.1 Syntax 4-271
 - 4.2.29.1.2 Parameters 4-271
 - 4.2.29.1.3 Description 4-271
 - 4.2.29.1.4 Return Values 4-271
- 4.2.30 IdmaObjectFactory 4-271
 - 4.2.30.1 IdmaObjectFactory::CreateObject 4-271
 - 4.2.30.1.1 Syntax 4-271

4.2.30.1.2	Parameters	4-272
4.2.30.1.3	Description	4-272
4.2.30.1.4	Return Values	4-272
4.2.31	IdmaOIID	4-273
4.2.31.1	IdmaOIID::GetDocSpaceId	4-273
4.2.31.1.1	Syntax	4-273
4.2.31.1.2	Parameters	4-273
4.2.31.1.3	Description	4-273
4.2.31.1.4	Return Values	4-273
4.2.31.2	IdmaOIID::GetGuid	4-273
4.2.31.2.1	Syntax	4-273
4.2.31.2.2	Parameters	4-274
4.2.31.2.3	Description	4-274
4.2.31.2.4	Return Values	4-274
4.2.31.3	IdmaOIID::GetObjectIdText	4-274
4.2.31.3.1	Syntax	4-274
4.2.31.3.2	Parameters	4-274
4.2.31.3.3	Description	4-274
4.2.31.3.4	Return Values	4-275
4.2.31.4	IdmaOIID::GetSystemId	4-275
4.2.31.4.1	Syntax	4-275
4.2.31.4.2	Parameters	4-275
4.2.31.4.3	Description	4-275
4.2.31.4.4	Return Values	4-275
4.2.32	IdmaProgressCallback	4-276
4.2.32.1	IdmaProgressCallback::ReportProgress	4-276
4.2.32.1.1	Syntax	4-276
4.2.32.1.2	Parameters	4-276
4.2.32.1.3	Description	4-276
4.2.32.1.4	Return Values	4-277
4.2.33	IdmaProperties	4-277
4.2.33.1	IdmaProperties::GetPropValBinaryById	4-278
4.2.33.1.1	Syntax	4-278
4.2.33.1.2	Parameters	4-278
4.2.33.1.3	Description	4-278
4.2.33.1.4	Return Values	4-278
4.2.33.2	IdmaProperties::GetPropValBinaryByIndex	4-278
4.2.33.2.1	Syntax	4-279
4.2.33.2.2	Parameters	4-279
4.2.33.2.3	Description	4-279
4.2.33.2.4	Return Values	4-279
4.2.33.3	IdmaProperties::GetPropValBooleanById	4-279
4.2.33.3.1	Syntax	4-280
4.2.33.3.2	Parameters	4-280
4.2.33.3.3	Description	4-280
4.2.33.3.4	Return Values	4-280
4.2.33.4	IdmaProperties::GetPropValBooleanByIndex	4-280
4.2.33.4.1	Syntax	4-280
4.2.33.4.2	Parameters	4-281

4.2.33.4.3	Description	4-281
4.2.33.4.4	Return Values	4-281
4.2.33.5	IdmaProperties::GetPropValDateTimeById	4-281
4.2.33.5.1	Syntax	4-281
4.2.33.5.2	Parameters	4-281
4.2.33.5.3	Description	4-282
4.2.33.5.4	Return Values	4-282
4.2.33.6	IdmaProperties::GetPropValDateTimeByIndex	4-282
4.2.33.6.1	Syntax	4-282
4.2.33.6.2	Parameters	4-282
4.2.33.6.3	Description	4-282
4.2.33.6.4	Return Values	4-283
4.2.33.7	IdmaProperties::GetPropValFloat64ById	4-283
4.2.33.7.1	Syntax	4-283
4.2.33.7.2	Parameters	4-283
4.2.33.7.3	Description	4-283
4.2.33.7.4	Return Values	4-284
4.2.33.8	IdmaProperties::GetPropValFloat64ByIndex	4-284
4.2.33.8.1	Syntax	4-284
4.2.33.8.2	Parameters	4-284
4.2.33.8.3	Description	4-284
4.2.33.8.4	Return Values	4-284
4.2.33.9	IdmaProperties::GetPropValIdById	4-285
4.2.33.9.1	Syntax	4-285
4.2.33.9.2	Parameters	4-285
4.2.33.9.3	Description	4-285
4.2.33.9.4	Return Values	4-285
4.2.33.10	IdmaProperties::GetPropValIdByIndex	4-286
4.2.33.10.1	Syntax	4-286
4.2.33.10.2	Parameters	4-286
4.2.33.10.3	Description	4-286
4.2.33.10.4	Return Values	4-286
4.2.33.11	IdmaProperties::GetPropValInteger32ById	4-286
4.2.33.11.1	Syntax	4-286
4.2.33.11.2	Parameters	4-287
4.2.33.11.3	Description	4-287
4.2.33.11.4	Return Values	4-287
4.2.33.12	IdmaProperties::GetPropValInteger32ByIndex	4-287
4.2.33.12.1	Support for this method is optional.	4-287
4.2.33.12.2	Parameters	4-287
4.2.33.12.3	Description	4-288
4.2.33.12.4	Return Values	4-288
4.2.33.13	IdmaProperties::GetPropValObjectById	4-288
4.2.33.13.1	Syntax	4-288
4.2.33.13.2	Parameters	4-288
4.2.33.13.3	Description	4-288
4.2.33.13.4	Return Values	4-289
4.2.33.14	IdmaProperties::GetPropValObjectByIndex	4-290
4.2.33.14.1	Syntax	4-290
4.2.33.14.2	Parameters	4-290

4.2.33.14.3 Description	4-290
4.2.33.14.4 Return Values	4-291
4.2.33.15 IdmaProperties::GetPropValStringById	4-291
4.2.33.15.1 Syntax	4-291
4.2.33.15.2 Parameters	4-291
4.2.33.15.3 Description	4-292
4.2.33.15.4 Return Values	4-292
4.2.33.16 IdmaProperties::GetPropValStringByIndex	4-292
4.2.33.16.1 Syntax	4-292
4.2.33.16.2 Parameters	4-292
4.2.33.16.3 Description	4-292
4.2.33.16.4 Return Values	4-293
4.2.34 IdmaRelationshipOrdering	4-293
4.2.34.1 IdmaRelationshipOrdering::SetOrdering	4-293
4.2.34.1.1 Syntax	4-293
4.2.34.1.2 Parameters	4-293
4.2.34.1.3 Description	4-294
4.2.34.1.4 Return Values	4-295
4.2.35 IdmaResultSet	4-295
4.2.35.1 IdmaResultSet::GetNextResultRow	4-295
4.2.35.1.1 Syntax	4-295
4.2.35.1.2 Parameters	4-295
4.2.35.1.3 Description	4-295
4.2.35.1.4 Return Values	4-296
4.2.35.2 IdmaResultSet::IsResultReady	4-296
4.2.35.2.1 Syntax	4-296
4.2.35.2.2 Parameters	4-296
4.2.35.2.3 Description	4-296
4.2.35.2.4 Return Values	4-297
4.2.35.3 IdmaResultSet::ReExecuteQuery	4-297
4.2.35.3.1 Syntax	4-297
4.2.35.3.2 Parameters	4-297
4.2.35.3.3 Description	4-297
4.2.35.3.4 Return Values	4-298
4.2.35.4 IdmaResultSet::TerminateResults	4-298
4.2.35.4.1 Syntax	4-298
4.2.35.4.2 Description	4-298
4.2.35.4.3 Return Values	4-298
4.2.36 IdmaScope	4-298
4.2.36.1 IdmaScope::ExecuteSearch	4-298
4.2.36.1.1 Syntax	4-298
4.2.36.1.2 Parameters	4-299
4.2.36.1.3 Description	4-299
4.2.36.1.4 Return Values	4-301
4.2.36.2 IdmaScope::GetOperatorDescription	4-302
4.2.36.2.1 Syntax	4-302
4.2.36.2.2 Parameters	4-302
4.2.36.2.3 Description	4-302
4.2.36.2.4 Return Values	4-302

4.2.37 IdmaScopeFactory	4-302
4.2.37.1 IdmaScopeFactory::CreateScope	4-303
4.2.37.1.1 Syntax	4-303
4.2.37.1.2 Parameters	4-303
4.2.37.1.3 Description	4-303
4.2.37.1.4 Return Values	4-304
4.2.37.2 IdmaScopeFactory::CreateScopeList	4-304
4.2.37.2.1 Syntax	4-304
4.2.37.2.2 Parameters	4-304
4.2.37.2.3 Description	4-305
4.2.37.2.4 Return Values	4-305
4.2.38 IdmaServiceRegistry	4-305
4.2.38.1 IdmaServiceRegistry::FindNextRegistryElement	4-305
4.2.38.1.1 Syntax	4-305
4.2.38.1.2 Parameters	4-306
4.2.38.1.3 Description	4-307
4.2.38.1.4 Return Values	4-307
4.2.38.2 IdmaServiceRegistry::GetRegistryElement	4-307
4.2.38.2.1 Syntax	4-308
4.2.38.2.2 Parameters	4-308
4.2.38.2.3 Description	4-308
4.2.38.2.4 Return Values	4-308
4.2.38.3 IdmaServiceRegistry::PublishRegistryElement	4-309
4.2.38.3.1 Syntax	4-309
4.2.38.3.2 Parameters	4-309
4.2.38.3.3 Description	4-309
4.2.38.3.4 Return Values	4-309
4.2.38.4 IdmaServiceRegistry::RegisterServiceElement	4-309
4.2.38.4.1 Syntax	4-310
4.2.38.4.2 Parameters	4-310
4.2.38.4.3 Description	4-310
4.2.38.4.4 Return Values	4-311
4.2.38.5 IdmaServiceRegistry::RemoveRegistryElement	4-311
4.2.38.5.1 Syntax	4-311
4.2.38.5.2 Parameters	4-311
4.2.38.5.3 Description	4-311
4.2.38.5.4 Return Values	4-312
4.2.38.6 IdmaServiceRegistry::UnpublishRegistryElement	4-312
4.2.38.6.1 Syntax	4-312
4.2.38.6.2 Parameters	4-312
4.2.38.6.3 Description	4-312
4.2.38.6.4 Return Values	4-312
4.2.39 IdmaStream	4-313
4.2.39.7 IdmaStream::ReadStreamData	4-313
4.2.39.7.1 Syntax	4-313
4.2.39.7.2 Parameters	4-313
4.2.39.7.3 Description	4-313
4.2.39.7.4 Return Values	4-313
4.2.39.8 IdmaStream::SetStreamPosition	4-313

4.2.39.8.1 Syntax	4-314
4.2.39.8.2 Parameters	4-314
4.2.39.8.3 Description	4-314
4.2.39.8.4 Return Values	4-314
4.2.40 IdmaSystem	4-315
4.2.40.1 IdmaSystem::ConnectDocSpace	4-315
4.2.40.1.1 Syntax	4-315
4.2.40.1.2 Parameters	4-315
4.2.40.1.3 Description	4-315
4.2.40.1.4 Return Values	4-316
4.2.40.2 IdmaSystem::EnumerateDocSpaces	4-316
4.2.40.2.1 Syntax	4-316
4.2.40.2.2 Parameters	4-316
4.2.40.2.3 Description	4-316
4.2.40.2.4 Return Values	4-317
4.2.40.3 IdmaSystem::GetResultCodeDescription	4-317
4.2.40.3.1 Syntax	4-317
4.2.40.3.2 Parameters	4-317
4.2.40.3.3 Description	4-317
4.2.40.3.4 Return Values	4-318
4.2.41 IdmaSystemManager	4-318
4.2.41.1 IdmaSystemManager::ConnectSystem	4-318
4.2.41.1.1 Syntax	4-318
4.2.41.1.2 Parameters	4-318
4.2.41.1.3 Description	4-318
4.2.41.1.4 Return Values	4-319
4.2.41.2 IdmaSystemManager::EnumerateSystems	4-319
4.2.41.2.1 Syntax	4-319
4.2.41.2.2 Parameters	4-319
4.2.41.2.3 Description	4-319
4.2.41.2.4 Return Values	4-320
4.2.41.3 IdmaSystemManager::GetMalloc	4-320
4.2.41.3.1 Syntax	4-320
4.2.41.3.2 Parameters	4-320
4.2.41.3.3 Description	4-321
4.2.41.3.4 Return Values	4-321
4.2.42 IdmaTextOrdering	4-321
4.2.42.1 IdmaTextOrdering::CompareStrings	4-321
4.2.42.1.1 Syntax	4-321
4.2.42.1.2 Parameters	4-322
4.2.42.1.3 Description	4-322
4.2.42.1.4 Return Values	4-322
4.2.43 IdmaVersionable	4-322
4.2.43.1 IdmaVersionable::SetCheckIn	4-322
4.2.43.1.1 Syntax	4-322
4.2.43.1.2 Parameters	4-323
4.2.43.1.3 Description	4-323
4.2.43.1.4 Return Values	4-324
4.2.43.1.5 Deferred Return Values	4-324

4.2.44	IdmaVersionSeries	4-324
4.2.44.1	IdmaVersionSeries::SetCheckOutNext	4-324
4.2.44.1.1	Syntax	4-324
4.2.44.1.2	Parameters	4-325
4.2.44.1.3	Description	4-325
4.2.44.1.4	Return Values	4-326
4.2.44.1.5	Deferred Return Values	4-326
4.2.44.2	IdmaVersionSeries::SetReserveNext	4-326
4.2.44.2.1	Syntax	4-326
4.2.44.2.2	Parameters	4-326
4.2.44.2.3	Description	4-326
4.2.44.2.4	Return Values	4-327
4.2.44.2.5	Deferred Return Values	4-327
4.2.44.3	IdmaVersionSeries::SetRevoke	4-327
4.2.44.3.1	Syntax	4-328
4.2.44.3.2	Description	4-328
4.2.44.3.3	Return Values	4-328
4.2.44.3.4	Deferred Return Values	4-328
4.2.45	IUnknown	4-328
4.2.45.1	IUnknown::AddRef	4-328
4.2.45.1.1	Syntax	4-328
4.2.45.1.2	Description	4-328
4.2.45.2	IUnknown::QueryInterface	4-329
4.2.45.2.1	Syntax	4-329
4.2.45.2.2	Parameters	4-329
4.2.45.2.3	Description	4-329
4.2.45.2.4	Return Values	4-329
4.2.45.3	IUnknown::Release	4-329
4.2.45.3.1	Syntax	4-329
4.2.45.3.2	Description	4-330
4.3	Macros	4-331
4.3.1	DMA_CREATE_STRING	4-331
4.3.1.1	Syntax	4-331
4.3.1.2	Parameters	4-331
4.3.1.3	Description	4-331
4.3.2	DMA_FREE_STRING	4-332
4.3.2.1	Syntax	4-332
4.3.2.2	Parameters	4-332
4.3.2.3	Description	4-332
4.3.3	DMA_GET_STRING_CHAR_COUNT	4-332
4.3.3.1	Syntax	4-332
4.3.3.2	Parameters	4-332
4.3.3.3	Description	4-332
4.3.4	DMA_GET_STRING_TEXT	4-333
4.3.4.1	Syntax	4-333
4.3.4.2	Parameters	4-333
4.3.4.3	Description	4-333

4.4 DMA Return Codes 4-334

4.5 Join Operators 4-339

4.5.1 Joins and Three Valued Elimination 4-339

4.5.2 Cross Join 4-340

4.5.3 Inner Join 4-340

4.5.4 Left Outer Join 4-340

4.5.5 Right Outer Join 4-340

4.5.6 DMA Defined Join Operators 4-341

4.6 DMA Well Known Query Operators 4-342

4.6.1 Ordering Rules 4-342

4.6.2 DMA Well Known Query Operators Descriptions 4-342

4.6.2.1 dmaQueryOperator_And 4-345

4.6.2.2 dmaQueryOperator_Or 4-345

4.6.2.3 dmaQueryOperator_Not 4-345

4.6.2.4 dmaQueryOperator_EqualBinary 4-346

4.6.2.5 dmaQueryOperator_UnequalBinary 4-346

4.6.2.6 dmaQueryOperator_GreaterBinary 4-346

4.6.2.7 dmaQueryOperator_GreaterOrEqualBinary 4-346

4.6.2.8 dmaQueryOperator_LessBinary 4-346

4.6.2.9 dmaQueryOperator_LessOrEqualBinary 4-346

4.6.2.10 dmaQueryOperator_EqualString 4-346

4.6.2.11 dmaQueryOperator_UnequalString 4-346

4.6.2.12 dmaQueryOperator_GreaterString 4-346

4.6.2.13 dmaQueryOperator_GreaterOrEqualString 4-346

4.6.2.14 dmaQueryOperator_LessString 4-346

4.6.2.15 dmaQueryOperator_LessOrEqualString 4-346

4.6.2.16 dmaQueryOperator_EqualBoolean 4-347

4.6.2.17 dmaQueryOperator_UnequalBoolean 4-347

4.6.2.18 dmaQueryOperator_EqualInteger32 4-347

4.6.2.19 dmaQueryOperator_UnequalInteger32 4-347

4.6.2.20 dmaQueryOperator_GreaterInteger32 4-347

4.6.2.21 dmaQueryOperator_GreaterOrEqualInteger32 4-347

4.6.2.22 dmaQueryOperator_LessInteger32 4-347

4.6.2.23 dmaQueryOperator_LessOrEqualInteger32 4-347

4.6.2.24 dmaQueryOperator_EqualFloat64 4-347

4.6.2.25 dmaQueryOperator_UnequalFloat64 4-347

4.6.2.26 dmaQueryOperator_GreaterFloat64 4-347

4.6.2.27 dmaQueryOperator_GreaterOrEqualFloat64 4-347

4.6.2.28 dmaQueryOperator_LessFloat64 4-348

4.6.2.29 dmaQueryOperator_LessOrEqualFloat64 4-348

4.6.2.30 dmaQueryOperator_EqualDateTime 4-348

4.6.2.31 dmaQueryOperator_UnequalDateTime 4-348

4.6.2.32 dmaQueryOperator_GreaterDateTime 4-348

4.6.2.33 dmaQueryOperator_GreaterOrEqualDateTime 4-348

4.6.2.34	dmaQueryOperator_LessDateTime	4-348
4.6.2.35	dmaQueryOperator_LessOrEqualDateTime	4-348
4.6.2.36	dmaQueryOperator_EqualId	4-348
4.6.2.37	dmaQueryOperator_UnequalId	4-348
4.6.2.38	dmaQueryOperator_EqualObject	4-348
4.6.2.39	dmaQueryOperator_IsDefined	4-349
4.6.2.40	dmaQueryOperator_IsNull	4-349
4.6.2.41	dmaQueryOperator_Like	4-349
4.6.2.42	dmaQueryOperator_IsClass	4-349
4.6.2.43	dmaQueryOperator_AddInteger32	4-349
4.6.2.44	dmaQueryOperator_SubtractInteger32	4-349
4.6.2.45	dmaQueryOperator_NegateInteger32	4-349
4.6.2.46	dmaQueryOperator_AbsoluteValueInteger32	4-350
4.6.2.47	dmaQueryOperator_MultiplyInteger32	4-350
4.6.2.48	dmaQueryOperator_DivideInteger32	4-350
4.6.2.49	dmaQueryOperator_AddFloat64	4-350
4.6.2.50	dmaQueryOperator_SubtractFloat64	4-350
4.6.2.51	dmaQueryOperator_NegateFloat64	4-350
4.6.2.52	dmaQueryOperator_AbsoluteValueFloat64	4-350
4.6.2.53	dmaQueryOperator_MultiplyFloat64	4-350
4.6.2.54	dmaQueryOperator_DivideFloat64	4-350
4.6.2.55	dmaQueryOperator_Integer32ToFloat32	4-350
4.6.2.56	dmaQueryOperator_Float64ToInteger32Round	4-350
4.6.2.57	dmaQueryOperator_Float64ToInteger32Truncate	4-350
4.6.2.58	dmaQueryOperator_Exists	4-351
4.6.2.59	dmaQueryOperator_InBinary	4-351
4.6.2.60	dmaQueryOperator_InBoolean	4-351
4.6.2.61	dmaQueryOperator_InString	4-351
4.6.2.62	dmaQueryOperator_InInteger32	4-351
4.6.2.63	dmaQueryOperator_InFloat64	4-351
4.6.2.64	dmaQueryOperator_InDateTime	4-352
4.6.2.65	dmaQueryOperator_InId	4-352
4.7 DMA Property DataTypes 4-353		
4.8 Conformance 4-354		
4.8.1 Mandatory Baseline 4-354		
4.8.1.1 General 4-354		
4.8.1.1.1 System Metadata Spaces 4-355		
4.8.1.1.2 Individual Document Space Metadata Spaces 4-355		
4.8.1.1.3 Individual Document Space Scope Metadata Spaces 4-355		
4.8.1.1.4 Merged Scope Metadata Spaces 4-356		
4.8.1.1.5 Query Result Metadata Spaces 4-356		
4.8.1.2 System Object 4-356		
4.8.1.3 Document Repository 4-356		
4.8.2 Optional Capabilities 4-357		
4.8.2.1 Document Space Object 4-357		

4.8.2.2	General	4-357
4.8.2.2.1	DMAC_LEVEL	4-357
4.8.2.2.2	DMAC_AUTHENTICATE	4-357
4.8.2.2.3	DMAC_WRITEABLE_DOCSPACE	4-357
4.8.2.2.4	DMAC_CONNECT_VIA_OIID	4-358
4.8.2.3	Content	4-358
4.8.2.3.1	DMAC_CONTENT	4-358
4.8.2.3.2	DMAC_CONTENT_MULT_RENDITIONS	4-358
4.8.2.3.3	DMAC_CONTENT_MULT_ELEMENTS	4-358
4.8.2.3.4	DMAC_CONTENT_MIME_RENDITIONS	4-358
4.8.2.3.5	DMAC_CONTENT_UPDATEABLE	4-358
4.8.2.3.6	DMAC_CONTENT_CAPTURE_RESOURCE	4-359
4.8.2.3.7	DMAC_CONTENT_COPY_TO_RESOURCE	4-359
4.8.2.4	Containment	4-359
4.8.2.4.1	DMAC_CONTAIN_DIR_REQUIRED	4-359
4.8.2.4.2	DMAC_CONTAIN_REF_REQUIRED	4-359
4.8.2.4.3	DMAC_CONTAIN	4-360
4.8.2.4.4	DMAC_CONTAIN_REF	4-360
4.8.2.4.5	DMAC_CONTAIN_REF_CONTAINER	4-360
4.8.2.4.6	DMAC_CONTAIN_REF_CFG_HISTORY	4-360
4.8.2.4.7	DMAC_CONTAIN_REF_VER_SERIES	4-361
4.8.2.4.8	DMAC_CONTAIN_REF_VERS_DESC	4-361
4.8.2.4.9	DMAC_CONTAIN_REF_DOC_VER	4-361
4.8.2.4.10	DMAC_CONTAIN_REF_OTHER	4-361
4.8.2.4.11	DMAC_CONTAIN_REF_MAX_DEPTH	4-361
4.8.2.4.12	DMAC_CONTAIN_REF_ORDER	4-361
4.8.2.4.13	DMAC_CONTAIN_REF_ORDER_HEAD	4-362
4.8.2.4.14	DMAC_CONTAIN_REF_ORDER_HEAD_BEFORE	4-362
4.8.2.4.15	DMAC_CONTAIN_REF_ORDER_HEAD_AFTER	4-362
4.8.2.4.16	DMAC_CONTAIN_REF_ORDER_HEAD_FIRST	4-362
4.8.2.4.17	DMAC_CONTAIN_REF_ORDER_HEAD_LAST	4-362
4.8.2.4.18	DMAC_CONTAIN_REF_ORDER_HEAD_NO_CH	4-363
4.8.2.4.19	DMAC_CONTAIN_REF_ORDER_TAIL	4-363
4.8.2.4.20	DMAC_CONTAIN_REF_ORDER_TAIL_BEFORE	4-363
4.8.2.4.21	DMAC_CONTAIN_REF_ORDER_TAIL_AFTER	4-363
4.8.2.4.22	DMAC_CONTAIN_REF_ORDER_TAIL_FIRST	4-363
4.8.2.4.23	DMAC_CONTAIN_REF_ORDER_TAIL_LAST	4-364
4.8.2.4.24	DMAC_CONTAIN_REF_ORDER_TAIL_NO_CH	4-364
4.8.2.4.25	DMAC_CONTAIN_DIR	4-364
4.8.2.4.26	DMAC_CONTAIN_DIR_CONTAINER	4-364
4.8.2.4.27	DMAC_CONTAIN_DIR_CFG_HISTORY	4-365
4.8.2.4.28	DMAC_CONTAIN_DIR_VER_SERIES	4-365
4.8.2.4.29	DMAC_CONTAIN_DIR_VER_DESC	4-365
4.8.2.4.30	DMAC_CONTAIN_DIR_DOC_VER	4-365
4.8.2.4.31	DMAC_CONTAIN_DIR_OTHER	4-365
4.8.2.4.32	DMAC_CONTAIN_DIR_MAX_DEPTH	4-365
4.8.2.4.33	DMAC_CONTAIN_DIR_ORDER	4-366
4.8.2.4.34	DMAC_CONTAIN_DIR_ORDER_HEAD	4-366
4.8.2.4.35	DMAC_CONTAIN_DIR_ORDER_HEAD_BEFORE	4-366
4.8.2.4.36	DMAC_CONTAIN_DIR_ORDER_HEAD_AFTER	4-366
4.8.2.4.37	DMAC_CONTAIN_DIR_ORDER_HEAD_FIRST	4-366

- 4.8.2.4.38 DMAC_CONTAIN_DIR_ORDER_HEAD_LAST 4-367
- 4.8.2.4.39 DMAC_CONTAIN_DIR_ORDER_HEAD_NO_CH 4-367
- 4.8.2.4.40 DMAC_CONTAIN_DIR_ORDER_TAIL 4-367
- 4.8.2.4.41 DMAC_CONTAIN_DIR_ORDER_TAIL_BEFORE 4-367
- 4.8.2.4.42 DMAC_CONTAIN_DIR_ORDER_TAIL_AFTER 4-367
- 4.8.2.4.43 DMAC_CONTAIN_DIR_ORDER_TAIL_FIRST 4-368
- 4.8.2.4.44 DMAC_CONTAIN_DIR_ORDER_TAIL_LAST 4-368
- 4.8.2.4.45 DMAC_CONTAIN_DIR_ORDER_TAIL_NO_CH 4-368
- 4.8.2.5 Versioning 4-368
 - 4.8.2.5.1 DMAC_VERSION 4-369
 - 4.8.2.5.2 DMAC_VERSION_REQUIRED 4-369
 - 4.8.2.5.3 DMAC_VERSION_OTHER_ENTITIES 4-369
 - 4.8.2.5.4 DMAC_VERSION_CHECKIN_EXISTING 4-369
 - 4.8.2.5.5 DMAC_VERSION_THREADED 4-369
 - 4.8.2.5.6 DMAC_VERSION_CHECKOUT 4-369
 - 4.8.2.5.7 DMAC_VERSION_CHANGE_CURRENT 4-370
 - 4.8.2.5.8 DMAC_VERSION_DELETE_VERSION_DESCRIPTION 4-370
- 4.8.2.6 Query 4-370
 - 4.8.2.6.1 DMAC_QUERY 4-370
 - 4.8.2.6.2 DMAC_QUERY_EXPRESSION 4-371
 - 4.8.2.6.3 DMAC_QUERY_JOINS 4-371
 - 4.8.2.6.4 DMAC_QUERY_ORDERINGS 4-371
 - 4.8.2.6.5 DMAC_3V_ELIMINATION 4-371
 - 4.8.2.6.6 DMAC_LOCKING 4-371
 - 4.8.2.6.7 DMAC_LOCKING_WRITER 4-372
 - 4.8.2.6.8 DMAC_LOCKING_READER 4-372
 - 4.8.2.6.9 DMAC_LOCKING_EXISTENCE 4-372

5 DMA Appendix 5-1

5.1 System & Document Space Connection Example 5-2

5.2 Content Examples 5-7

- 5.2.1 CTEDIT.CPP Example File 5-7
- 5.2.2 CTNEWDOC.CPP Example File 5-12
- 5.2.3 CTGETDOC.CPP Example File 5-15

5.3 Pretty Print Example 5-18

5.4 Query Examples 5-24

- 5.4.1 query.h 5-25
- 5.4.2 query.cpp 5-25
- 5.4.3 quexp1.cpp 5-25
- 5.4.4 QUERY.H Example File 5-26
- 5.4.5 QUERY.CPP Example File 5-29
- 5.4.6 QUERY1.CPP Example File 5-43
- 5.4.7 QUERY2.CPP Example File 5-47
- 5.4.8 QUERY3.CPP Example File 5-51

5.5 Containment Examples 5-55

- 5.5.1 COSAMPLE1.CPP Example File 5-55
- 5.5.2 COSAMPLE2.CPP Example File 5-57

5.6 Versioning Example 5-65

5.7 DMA Glossary 5-69

List of Figures

Figure 1-1: UML Diagramming Key	1-2
Figure 2-1: DMA Uniform Access Model	2-5
Figure 2-2: Application Architecture Utilizing DMA	2-7
Figure 3-1: DMA Class Hierarchy	3-4
Figure 3-2: DMA Class and Property Description Classes	3-6
Figure 3-3: DMA Integration Model Components	3-32
Figure 3-4: DMA Integration Model and Localization Considerations	3-37
Figure 3-5: Basic Client-Server Distribution Model	3-52
Figure 3-6: Basic Point-of-Presence Model	3-52
Figure 3-7: Varieties of Distribution at a Point-of-Access	3-54
Figure 3-8: Creation of a Document Space Scope object	3-56
Figure 3-9: Execution of a Query	3-59
Figure 3-10: Example "From Expression"	3-66
Figure 3-11: Example Query Object	3-72
Figure 3-12: Creation of a Merged Scope Object	3-78
Figure 3-13: Objects Involved With Merged Scope	3-80
Figure 3-14: Cases Where Partial Ordering Cannot Be Preserved	3-83
Figure 3-15: DMA Class Inheritance Hierarchy for Containment	3-94
Figure 3-16: Referential Containment Relationship Object Example	3-97
Figure 3-17: Direct Containment Relationship Object Example	3-97
Figure 3-18: Sample DMA document	3-107
Figure 3-19: DMA Content Class Inheritance Diagram	3-113
Figure 3-20: DMA Model of Versioned Objects (conceptual)	3-118
Figure 3-21: DMA Version Management Navigation Model	3-119
Figure 3-22: DMA Version Management Inheritance Diagram	3-120
Figure 3-23: Linear Versioning	3-122
Figure 3-24: Branched Versioning	3-123
Figure 3-25: Threaded Versioning	3-124

1 Introduction

1.1 How to Use This Document

Welcome to the Document Management Alliance's DMA 1.0 Specification!
This specification has been approved by the DMA Advisory Council for general use, as of December 24, 1997.

DMA Technical Committee

Copyright © 1995, 1996, 1997 AllIM International, All rights reserved.

Revised: October 9, 1998

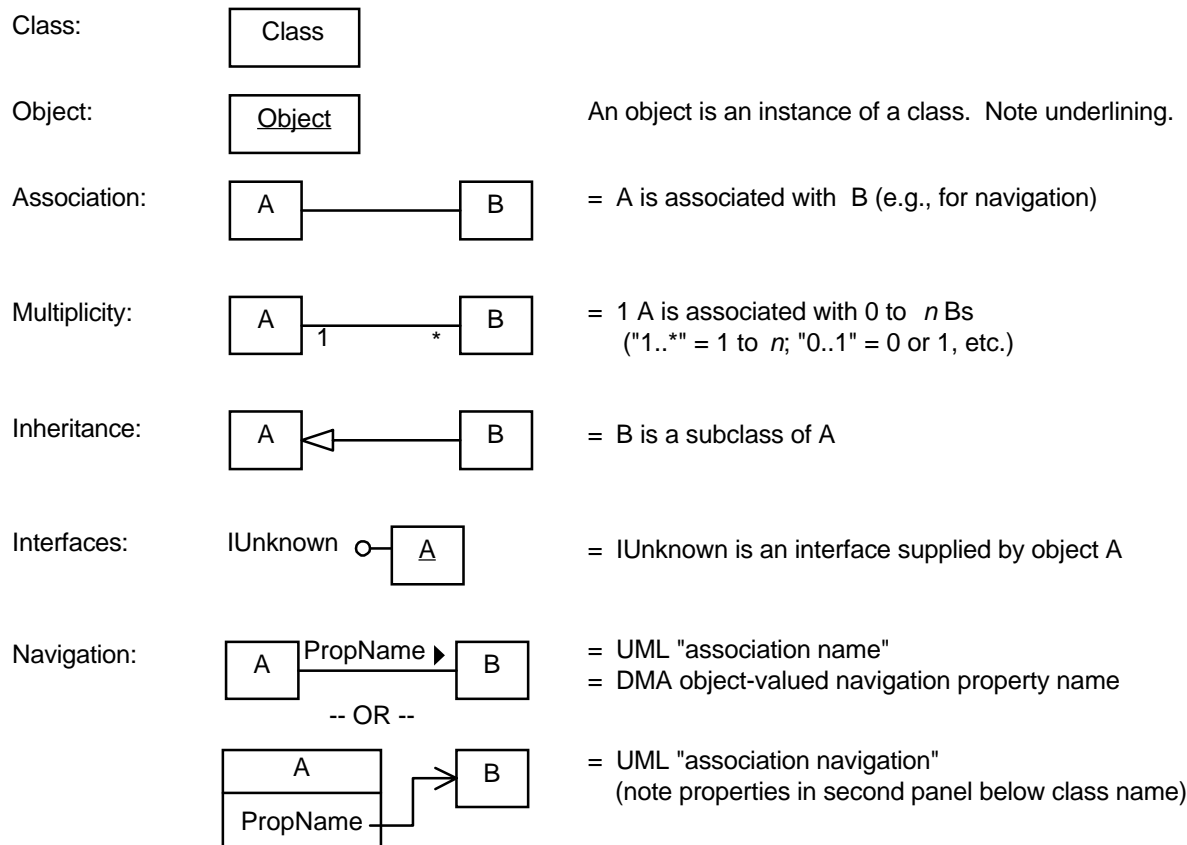
This specification enables interoperability between document management applications and systems. This document consists of four primary sections:

- Overview: Read this section to get a high level understanding of the DMA vision and approach to interoperability.
- DMA Architecture: Read this section to gain an understanding of the overall DMA architecture.
- DMA Reference: Read this section to understand the details of the DMA object model, the set of COM interfaces defined by the architecture, and the definition of DMA conformance.
- DMA Appendix: Read this section to see DMA code samples, and the DMA Glossary.

1.1.1 Diagramming

This specification uses a subset of the Unified Modeling Language (UML) to depict objects, relationships and classes. Figure 1 explains the notations used in this specification. For more information on UML, visit <http://www.rational.com/uml>.

Key to Unified Modeling Language Notation (UML 1.0)



(This is a small subset of UML. See <http://www.rational.com/uml> for the full definition of UML.)

Figure 1-1: UML Diagramming Key

1.2 Objectives

The Document Management Alliance (DMA) is a task force of AIIM International dedicated to the realization of a uniform approach for creation and operation of enterprise-wide document-management systems. The primary product of DMA is a specification for an integration model and the interfaces by which applications and services from a rich variety of sources can be integrated into a document-management solution.

1.3 History

The AIIM DMA Task Force was created in April of 1995. It was formed as a result of a merger of two previous document management standards efforts - DEN and Shamrock. DMA milestones are listed below:

- April 1995. DMA created as AIIM Task Force
- November 1995. DEN/Shamrock Convergence Document published. This document represented a first pass at a unified architecture, consolidating work done in DEN and Shamrock. Also, scope of DMA 1.0 architecture specified.
- April 1996. DMA prototype at AIIM Show and Conference. Six member companies of the DMA Task Force cooperated to produce a working prototype of the draft DMA architecture described in the DEN/Shamrock Convergence Document
- September 1996. 0.75 version of DMA specification published, showing progress to date on DMA specification.
- February 1997. DMA 0.9 specification published. 6-9 month trial use begun.
- April 1997. DMA prototype at AIIM Show and Conference. DMA member companies cooperated to produce another working prototype, this time based on the 0.9 specification.
- September 1997. Trial use coverage completed.
- November 1997. DMA 1.0 specification published -- incorporating trial use refinements -- for discussion, review, and vote.
- December 1997. DMA 1.0 specification approved unanimously by the DMA Technical Committee and then by the DMA Advisory Council for general use.

1.4 Future Activities

Future DMA activities will be driven by market interest generated by vendors and users using the DMA 1.0 specification. At this time, we believe that future activities could include work in the following areas:

- Interoperability with other proposed standards (e.g., ODMA, WfMC, Web-DAV)
- Additional work on DMA/Internet interoperability (e.g., Java)
- Additional support for compound documents, content-based search, security, directory service integration, transactions, and/or audit trails.

1.5 Acknowledgments

The DMA Technical Committee wishes to acknowledge the support and encouragement of the following individuals and organizations:

- DENrock participants. Technical and marketing staff from IBM, Saros, Novell and Xerox/XSoft worked through many difficult technical, cultural and business issues to create the merger of DEN and Shamrock that created DMA.
- All the member companies of the AIIM DMA Task Force. Each of these companies understand that the growth and success of the document management market, for both vendors and users, is dependent on cooperative efforts such as the DMA Task Force.
- AIIM International. Within AIIM, the following staff members (past and present employees) have been critical to our success: Cheryl Chadwell, Judy Kilpatrick, Jeanette Rogers, Barbara Talbott, Jeff White, and Marilyn Wright.
- DMA member companies that have hosted DMA TC meetings or working sessions: Autodesk, EDS, FileNET, IBM, Interleaf, Novell, Saros, and Xerox.
- DMA member companies and their representatives that contributed software or expertise to live DMA demonstrations at the 1996 and 1997 AIIM Shows and trial use prototypes:
 - Eastman Software (including separate work done by Eastman Kodak, Vantage, and Wang Software before they combined) -- Diane Entner, Richard Hogg, Jerry Jaworski, Lasantha Jayasinghe, Rex Lin, Mark Orpin, Rita Rapoport
 - EDS -- John Klavarian
 - FileNET -- Tanmoy Dutta, Chuck Fay, Jim Green, Glenn Peterson, Michael Seaman, Craig Takenaga
 - Hyland Software -- John Loper
 - Interleaf -- George Florentine
 - Xerox -- Charlotte Baltus, Larry Bonham, Shuyuan Chen, Gary Gocek, Dennis Hamilton, Jim Mayer, Richard Sauvain, Jock Williams, Tom Wills
- DMA member companies and their representatives that participated actively in Technical Committee meetings in 1996 and 1997 to develop this specification:
 - Autodesk -- Myles Cagney

- Canon Information Systems -- Lee Farrell
- Documentum -- Jim Donahue
- Eastman Software (including separate work done by Eastman Kodak, Vantage, and Wang Software before they combined) -- Diane Entner, Richard Hogg, Lasantha Jayasinghe, Jerry Jaworski, Rex Lin, Mark Orpin, Todd Trimble
- FileNET -- Alan Babich, Chuck Fay, Dan Whelan
- Fuji Xerox -- Yuji Ikeda, Hiroaki Machida
- Fujitsu -- Ralph Ferris
- Hyland Software -- Alfonso Zubizarreta
- IBM -- Jay Unger
- Interleaf -- George Florentine
- Napersoft -- Tom Grannan, Scott Herter
- Novasoft -- Carl Theobald
- Oracle -- Fred Carter, John MacNaughton
- Saros (now FileNET) -- Bernard Chester, Jim Green, Glenn Peterson, Michael Seaman
- Toshiba -- Nobuhisa Yoda
- Xerox -- David Elliott, Dennis Hamilton, Richard Sauvain, Judith Slein, Jock Williams, Tom Wills
- DMA member companies and their representatives that participated actively in Advisory Council meetings in 1996 and 1997 to develop user requirements, promote DMA, and provide direction to the Technical Committee:
 - Aetna -- Mike Gotta
 - Boeing -- Benton Ong
 - DFR Technologies -- Jean-Fran³ois Tougard
 - Documentum -- Matt Shanahan
 - Eastman Software (including separate work done by Eastman Kodak, Vantage, and Wang Software before they combined) -- Diane Entner, Jerry Jaworski

- EDS -- Ernie Castillo
 - FileNET -- Lenny D'Amico, Mike Stiles, Ken Tarmas, Dan Whelan
 - Harper Collins -- Kevin Vaughn
 - Hewitt Associates LLC -- Dotty L. Posto
 - Hyland Software -- John Loper
 - Interleaf -- George Florentine
 - Myers-Tierney Associates -- Linda Myers-Tierney
 - Napersoft -- Tom Grannan
 - U.S. Department of Justice -- Dan Schneider
 - Xerox -- David Elliott, Dennis Hamilton, Jean Derick Heminway
 - Xplor International -- Toby Cobrin
- WebDAV Chair Jim Whitehead, for his participation in DMA Technical Committee meetings to facilitate synergy between WebDAV and DMA.
 - FileNET Technical Writer John Kinsky, for all his editorial suggestions improving the readability and clarity of the final specification and for his creation of the PDF rendition of this specification.

2 Overview

The Document Management Alliance (DMA) is an AIIM Task Force with the charter to develop a uniform programming model enabling enterprise-wide interoperability among document-oriented application programs and document management systems (DM systems) from different vendors. The primary product of DMA is a specification for an integration model and interfaces by which applications and services from a rich variety of sources can integrate into a document-management solution. The members of DMA include a diverse group of DMS vendor companies, end user companies, governmental agencies, industry analysts and consultants, and industry press.

2.1 The DMA Vision

The DMA Task Force and the DMA architecture exist because of a shared vision among users and vendors of document management systems. The DMA vision is best described as a software architecture that allows unification of all document management systems and document-aware application programs in an enterprise, regardless of vendor, hardware platform, or software platform, into one seamless document management system spanning the enterprise. This vision rewards users with uniform access to any document, any format, anywhere across an enterprise, despite the existence of "islands of information" -- that is, despite separate departmental document management systems and document-aware applications from different vendors which do not work together in the absence of the unifying architecture specified by DMA.

The DMA vision is realized in this specification as an object-oriented programming framework that document management vendors, integrators, and developers can use to provide their customers and users with the following capabilities:

- Uniform access to any document, anywhere in an enterprise
- Self-describing systems and documents for ease of setup and configuration
- Scalable document management solutions from legacy systems to fully-featured, state-of-the-art document management systems
- Expanded collaboration opportunities
- High-level integration of services and applications

2.1.1 Uniform Access to Documents Anywhere

"Uniformity of access" means an end-user need only learn a single application to access all documents in the enterprise. DMA frees an end-user from having to install, configure, master, and maintain a complex set of application programs from all the vendors of document management systems in an enterprise. DMA's unifying programming model allows a single application program to work with all DMA-compliant systems. End-users with DMA-enabled document management systems can perform the following actions:

- Build and use document management systems that provide uniformity of access and integration with other systems over a wide range of scales, from desktop to department to enterprise and beyond to federations of systems among strategically-linked organizations.
- Locate and use electronic documents, wherever they are and in whatever form they exist, while still operating local document-management systems designed around specific areas of practice and strategic business applications. Other organizations can access the local document-management environment for appropriate collaborative shared-use of documents.
- Retrieve a format appropriate for the user's machine. DMA provides the mechanism to track the document format, so that the appropriate application can be launched on retrieval, if available. If a user lacks the software package used to create the document originally, DMA allows a document management system to maintain additional "renditions" of the same material in alternate formats that are compatible with the user's installed software.
- Retain access to the document legacy of an enterprise over time. Documents remain accessible and usable in the face of technology substitutions, organizational and application growth, and changes of scale, technology, and distribution of document management in the enterprise, so long as the systems used are DMA-compliant.
- Freely choose best of breed DMA-compliant document management products with the assurance they will all work together. Innovations in document-centered applications, in document management systems and in document services can be rapidly deployed and integrated into enterprise information systems. (New capabilities can be smoothly introduced and quickly made effective.)

2.1.2 Self-describing Systems and Documents

DMA systems are self-describing in a way that captures the essential information about the systems and the electronic documents they manage. DMA provides for run-time discovery of systems, classes of documents, properties of those documents, and supported search operators for each property. DMA applications require no advance knowledge of classes of documents, proper-

ties, or search capabilities within each DM system. This approach simplifies the setup, configuration, maintenance, and overall complexity of these applications, which reduces the total cost of ownership per user.

DMA also provides a run-time "capabilities" mechanism for determining the set of optional DMA features offered by a particular DM system. This mechanism allows DMA applications to adjust automatically at run-time to the different feature sets offered by each DM system in an enterprise, thereby promoting interoperability and flexibility in the mix of DM products in an enterprise.

2.1.3 Scalable Solutions

DMA defines a scalable and flexible Application Programming Interface (API) that accommodates a simple property database or file-system-based storage repository all the way up to high-end, fully-featured document management systems. The scalability and flexibility of the API allow it to be used by vendors, integrators, or developers to integrate legacy DM systems of varying capabilities with state of the art DMA-compliant DM systems and DMA applications in a uniform and cohesive enterprise-wide DM system. Therefore, DMA can be used to protect and leverage the user's existing DM system investment and ease the conversion from legacy DM systems to new DMA-compliant DM systems.

2.1.4 Expanded Collaboration Opportunities

As the recent explosion in use of the World Wide Web has shown, users want to share and manipulate documents regardless of their location. DMA expands the opportunities for collaboration by bridging the "*islands of information*" and allowing users to access documents across previously separate organizations.

2.1.5 Higher-Level Integration of Services and Applications

Enterprise applications involve more than documents, even when documents are the principal carrier of the information being used. Document management systems are increasingly being used as services by which applications are constructed for support of enterprise activities and strategic applications. The integration of workflow-management systems with document-management systems is typical of this trend. The value of document-management systems is extended by being able to offer them as building blocks for extensive, specialized applications, just as database management systems serve as building blocks for on-line transaction-processing applications and other processes.

DMA systems will support increased integration into higher-level applications in three ways:

- DMA's uniform application interface makes consistent access to document collections available to both document-management applications and applications that incorporate document access in their implementations.
- DMA provides a service integration model that allows additional services to be incorporated and accessed, where appropriate, by delivery in the guise of document management services.
- DMA's interface model allows for regulated introduction of extensions to basic DMA interfaces, so that the DMA model does not inhibit introduction of specialized functionality.

2.2 The DMA Approach

The DMA specification establishes an open software framework in which developers and customers can configure document management services and repositories that inter-operate across different platforms and systems.

2.3 DMA Concepts

DMA provides a global document system in which users gain access to collections of documents using a uniform model. (See the DMA Uniform Access Model figure below.)

DMA is analogous to a library and other systems having well-defined methods for storing and finding information. The access provided through a DMA system is similar to using a library system in the following ways:

- A library system consists of multiple libraries, both public and private. A *DMA System* is like a library system, made up of one or more *DMA document spaces*.
- A DMA document space is similar to a single library or collection. The features and capabilities of a library may be described in a printed guide available to its patrons. Similarly, clients of a document space can find out its features and capabilities from descriptive information associated with the document space.
- *DMA document version ("DocVersion") objects* are like the items that can be borrowed from the library (books, videos, CDs, audiotapes, etc.).
- The materials in the library have properties -- attributes that describe the media and features of the item (e.g. creation tool, format, date created, last modification date, author, etc.). DMA calls these attributes of documents *property values*.
- Materials can be checked out from a library, which keeps tracks of who checked out each item and when. DMA defines a mechanism for a docu-

ment space to track this kind of “*document management state information*” for each document.

- A particular book may be available from a library in more than one rendering, for instance in printed form as well as audio form. A DMA document can similarly have multiple *Rendition objects*, for example one in HTML format and another in PDF format.
- In order to locate something, the user must first decide which library or collection (document space) is most likely to contain that item, then search a card catalog or computer catalog. DMA provides a powerful *search capability* for not only searching a single document space, but also across multiple document spaces in a System in a single query.
- The library user (DMA client) requests information from the librarian (DMA service provider). A document space implementation that responds to client requests is the prime example of a *DMA service provider*.
- The librarian (service provider) controls access to the library (security) and can find materials based on a variety of search criteria.

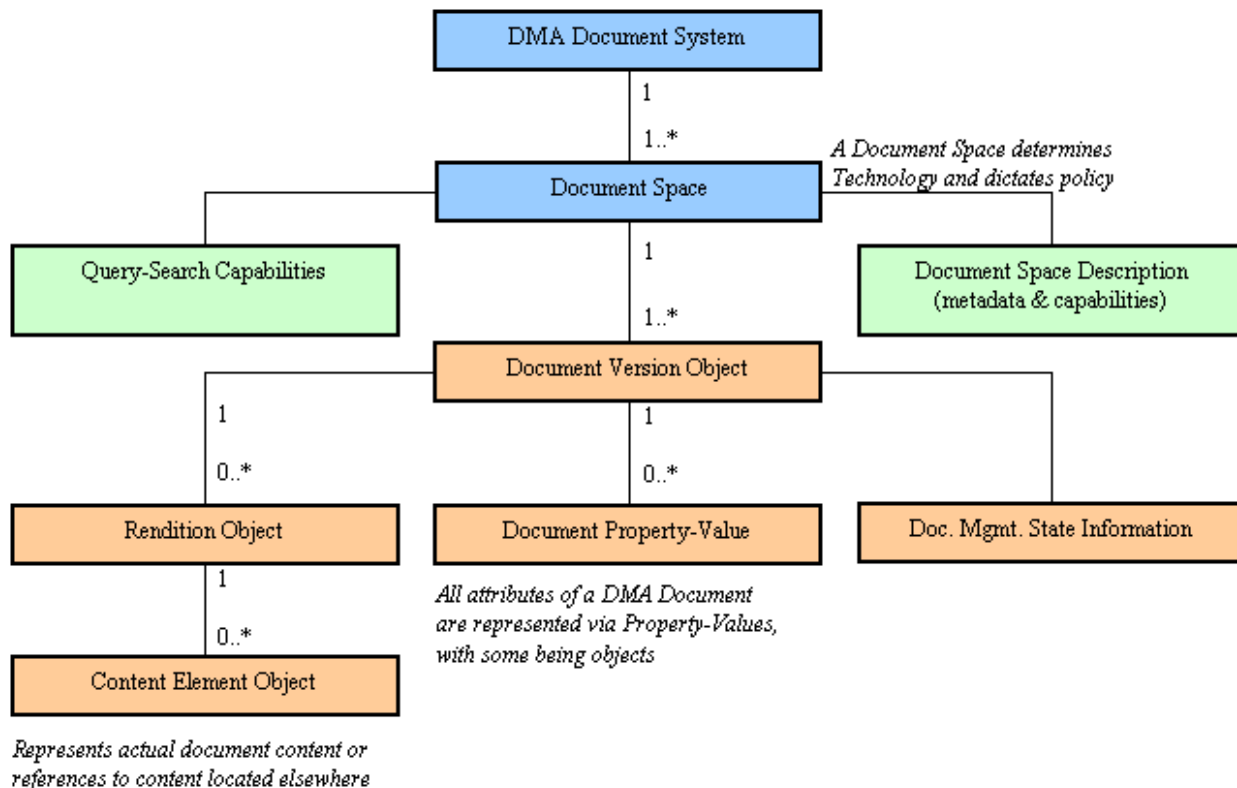


Figure 2-1: DMA Uniform Access Model

2.3.1 DMA Document Spaces

Within the document system, document spaces are the DMA representation of document collections. The DMA System provides a mechanism for an application to determine which spaces are available and what their capabilities are.

2.3.2 DMA Document Version Objects

Documents within the document space are described by properties (characteristics) of each individual document such as the application that was used to create the document, when the document was last accessed, etc.

2.4 Unification of Heterogeneous Document Management Systems

Because DMA systems provide common points of access to documents throughout an enterprise, users will be able to find and use documents created in most common office applications. Through a DMA system, users can have transparent, reliable, and uniform access to documents, regardless of their location or the form in which they exist. DMA defines the mechanism for a DM application to issue a single query to multiple DMA document collections to find documents that match that query.

2.4.1 DMA Application Architecture

The DMA application architecture is composed of three required elements:

- 1 DMA client applications
- 2 DMA middleware
- 3 DMA service providers

The figure below shows each of these elements.

DMA client applications interact with the DMA middleware and DMA system providers using the DMA Application Programming Interface (API). All user interactions and user interfaces are provided by the DMA client applications. These applications can be implemented in a variety of programming languages. The applications may combine other complementary document management specifications such as ODMA, the higher-level Open Document Management Interface, with DMA to achieve the benefits of both.

The *DMA middleware* provides the two primary functions of a DMA System, namely 1) cross-document space query coordination and 2) registration of DMA service providers so that they can be made available to client applications. The middleware, which gets its name from its position in the middle between DMA clients and service providers, interacts with the service providers using the same DMA API used by DMA clients, with the addition of a small set

of methods used by service providers to register themselves with the middle-ware.

The DMA service providers typically provide the services of a single library service or document management repository, termed a "document space" in DMA. In this figure, the Library Services and Value Added Services are DMA service providers. The Library Services represent specialized DMA-compliant document repositories from document management vendors. Examples of value added services defined by DMA 1.0 are merged scope plug-in, text ordering service elements and OIID parser service elements.

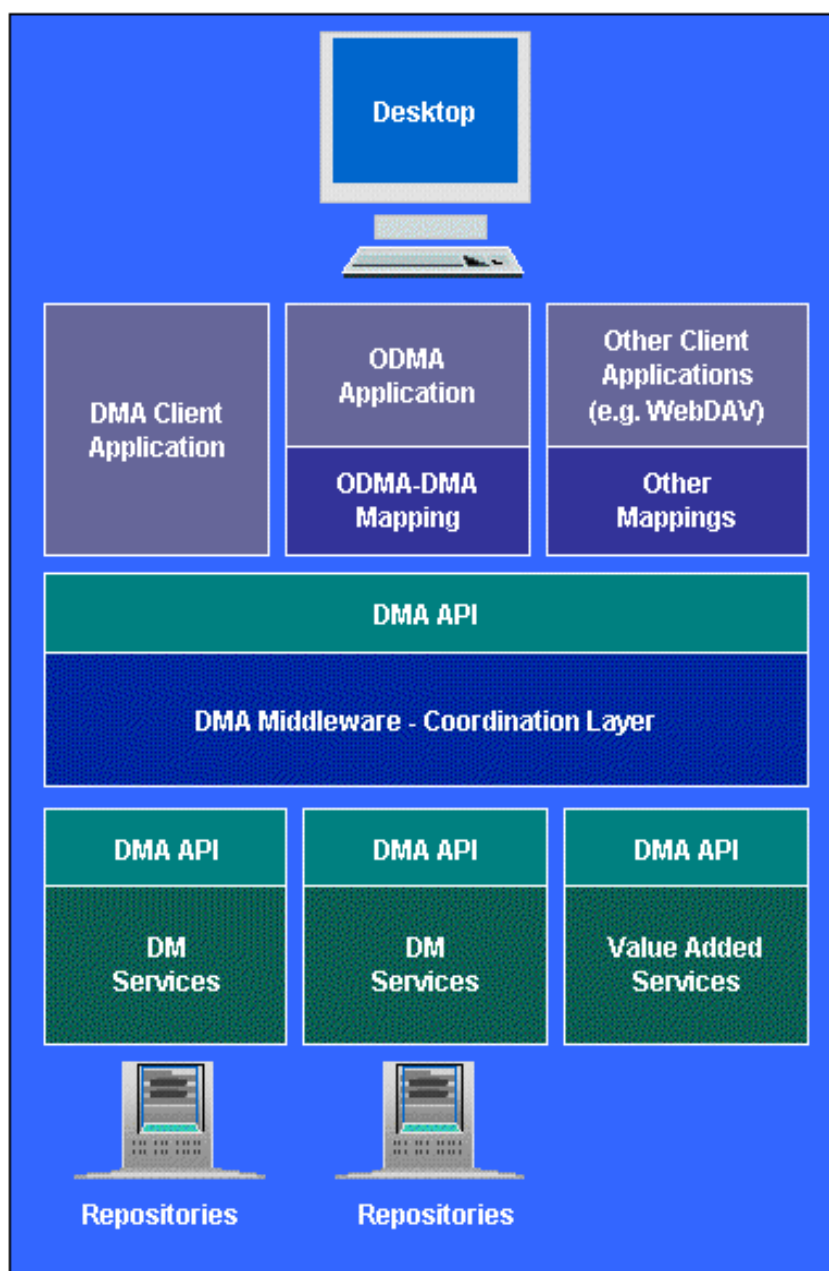


Figure 2-2: Application Architecture Utilizing DMA

2.4.2 Provision for Uniformity and Unification

DMA supports unification of document-management systems in three ways:

- 1 The uniformity in which document-management features are delivered to applications through the DMA model enables DMA-compliant front-end applications to access any DMA-compliant service element, regardless of differences in supplier, location, or capabilities.
- 2 The uniformity in which document-management features are supplied to the DMA system by service providers enables a DMA-compliant service provider to offer itself for access to any number of front-end applications.
- 3 The DMA coordination layer (middleware) provides additional value by facilitating unification as well as uniformity. DMA systems that provide coordinated searching allow different document collections to be accessed as if unified in a single collection. The unification mechanism allows exploration of combined collections without the front-end application or the user having to be concerned about differences in implementations of the collections.

2.4.3 DMA Allows Evolution

Document-management systems that are integrated under DMA are not restricted to DMA-compliant access alone. For example, an existing document-management system (designed as a standalone system) can be integrated under DMA in evolutionary steps:

- 1 Existing applications can continue to operate with the DM system as they do presently.
- 2 The DM system can be upgraded to offer access to its collection as if it is a DMA-compliant service provider. This service provider need not offer all the functions of the standalone system. It is common to simply provide for DMA-compliant access in order to support wider use and interchange of documents, but rely on the existing applications for more-complex operations, such as enforcement of policies, change control and so on.
- 3 The DM system's integrated front end can be expanded to not only access and manage material of the integrated system, it can incorporate the DMA interface to access DMA-compliant collections from other sources.
- 4 Combinations of the two steps can be taken for the same integrated document-management system. (The capabilities offered and exploited this way can be grown over time.)

2.4.4 Evolution of DMA capabilities and DMA-Compliant Components

The DMA specification provides a component based drop-in model for introduction of new document collections and new DM components and capabilities. In addition, the model allows for document-management systems of different levels of capability and performance to coexist in a single DMA system. The goal of the model is to allow document-management systems to be highly scaleable and for individual document-management systems to evolve over time in response to customer requirements (and the capability of underlying technology). The DMA model establishes basic operations and common elements of all DMA-compliant document-management systems. Many of the elements are elective in ways that can vary between collections and can vary over time as document-management software is upgraded and reconfigured. In addition, there is provision for customization beyond the specific provisions of DMA, but in a way that is compatible with DMA and will not interfere with the evolution of the DMA specification.

3 DMA Architecture

This section contains detailed information on the following topics:

- DMA Object Model
- DMA Interface and Process Model
- DMA Integration Model
- DMA Distribution Model
- DMA Query Model
- DMA Containment Model
- DMA Content Model
- DMA Versioning Model
- DMA Internationalization and Localization Model

3.1 DMA Object Model

3.1.1 Introduction

The DMA object model provides a scaleable, robust data model that which allows the integration of a broad range of document management applications and services. Within the context of the DMA model, objects are abstractions of data and methods that operate on this data. This approach provides a uniform programming model which insulates client applications from the implementation details of the underlying document space and allows the same application to operate over document spaces provided by different vendors. This important characteristic of the DMA object model is referred to as *separation of interface from implementation*.

The principal characteristics of DMA objects are listed below:

- DMA objects expose *methods*, which are client-invocable commands that provide the only means of observing and changing the state of the object and thus of accessing the underlying service represented by the object. Different kinds of object support different methods, but some methods are common to several different kinds of objects and have similar effect when invoked upon them. This *polymorphism* enhances the uniformity of the programming model. The mechanism for method invocation is that of the Microsoft Component Object Model (COM), and is described in detail in the Interface and Process Model section.
- principal among the methods supported by **all** DMA objects are those for retrieving and setting *properties*. A property is a named value representing part of the state of the object. Many operations in DMA are accomplished simply by getting and setting properties, and there are relatively few operations which require invocation of non-property related methods.
- every DMA object belongs to a particular *class*, which determines the methods and properties supported by the object. Furthermore, the *class description* (itself an object) is available as a property of every DMA object. Thus, DMA objects are runtime self-describing, facilitating the development of general purpose applications which can accommodate and adjust to vendor-specific or future DMA-defined extensions to the DMA model.
- many DMA objects provide a representation of and indirect access to data retained by the underlying document space. Such *persistent objects* support methods which cause them to be loaded with a reflection of that data and which allow changes made via methods on the object to be reflected back into the persistent store. Other objects are simply facilitators in the overall programming model and are therefore *ephemeral*.

3.1.2 DMA Classes

A DMA class is a definition for set of objects having the same supported methods and properties. It should be noted that this use of the term “class” is somewhat different from its usage in other contexts (such as COM or C++) where class signifies implementation. In DMA, class signifies a type of object for which there can be many implementations.

3.1.2.1 Class Naming

A class is named by one or more globally unique identifiers (GUIDs). The initial identifier is assigned by the original author of the class definition (DMA, in the case of the classes specified in this document). Additional identifiers may be assigned later, with the purpose of creating a common *alias identifier* applicable to initially distinct classes which have been recognized as being semantically equivalent. This feature is important in operations such as query which can span multiple document spaces. DMA 1.0 does not define mechanisms for alias management, but it is not expected that aliases will be necessary for any of the classes defined by DMA.

It is illegal for two classes of an individual Document Space to have a common alias ID. Alias ID's for classes and properties are strictly for purposes of meta-data unification across distinct Document Spaces when building a merged scope.

It is illegal to simultaneously assign the same ID to both a class and a property, or to both a class and a query operator, etc.

3.1.2.2 Class Inheritance

DMA supports the definition of a new class by derivation from a single existing class. If a distinct class R derives its definition directly from another class Q then we say that Q is the *immediate superclass* of R. If Q in turn has immediate superclass P, then both P and Q are *superclasses* of R. We say that a class Y is a *subclass* of class X if either X is Y or if X is a superclass of Y. If X is a superclass of Y (and therefore is not Y) then we say that Y is a *proper subclass* of X. If X is the immediate superclass of Y then Y is an *immediate subclass* of X. (By definition, an immediate subclass is a proper subclass).

(Note that a class is a subclass of itself, but not a superclass of itself).

An immediate subclass *inherits* the characteristics of its immediate superclass in the following fashion:

- all of the methods supported by the superclass are supported by the subclass
- all the properties supported by the superclass are supported by the subclass

- The definition of the subclass may diverge from that of the superclass in the following ways:
- additional methods may be supported.
- additional properties may be supported.
- an inherited property which is optional in the superclass may be mandatory in the subclass (but not vice versa).
- the default value for an inherited property may differ in the subclass.
- the permitted values for the inherited property may be further constrained in the subclass. (New permitted values cannot be introduced).

The subclass **must** differ from its superclass in respect of naming. No identifier used to name the subclass may also be an identifier of the immediate superclass, or any of its superclasses.

Multiple (different) subclasses can be derived from single superclass, and each subclass can be further derived from (multiple times), thus creating a hierarchy of increasingly specialized classes. The root of this hierarchical inheritance tree is the DMA base class (dmaClass_DMA), which defines the properties and methods required of every DMA object. The following diagram shows the upper levels of the DMA inheritance tree, with the subclasses grouped into functional areas.

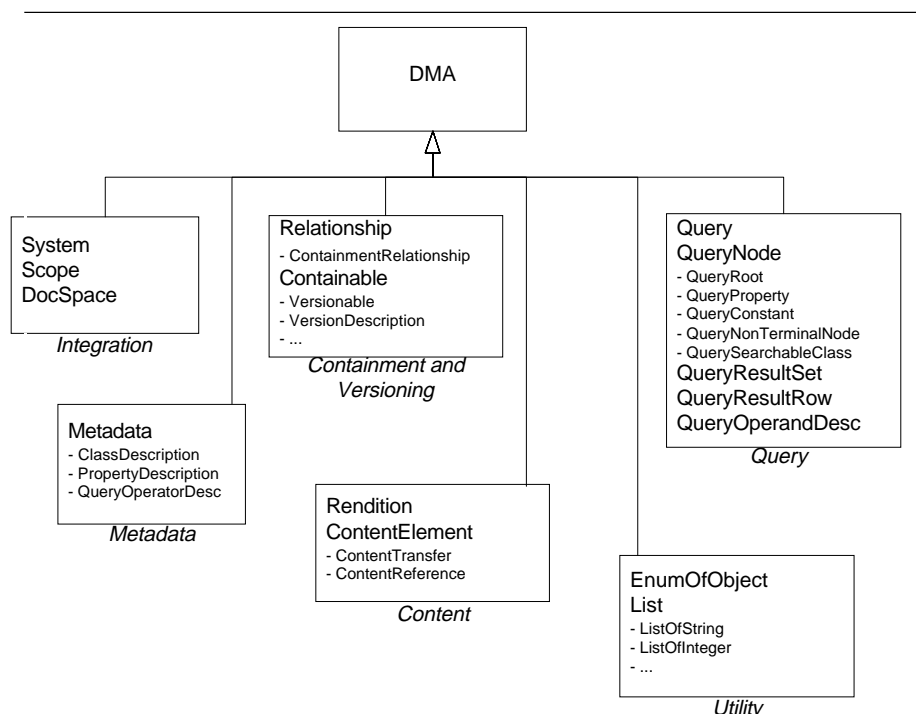


Figure 3-1: DMA Class Hierarchy

Every DMA object is an *instance* of exactly one class. However, the object may *conform* to many class definitions, since it possesses all the properties and supports all the methods required by each successive superclass of its class, all the way up to the base.

3.1.2.3 Class Metadata

Every DMA object describes the class to which it belongs by means of an object-valued property containing a description of the class.

The *metadata* object contained in the class description property is itself a DMA object, whose properties define the class. The metadata object includes the following:

- a multi-valued property defining the identifiers for the class
- a property containing the class description for the immediate superclass of the class, if any
- a property containing a list of the class descriptions for any subclasses of the class
- a property containing a list of descriptions of the properties of the class, including those inherited from the superclass. This is an ordered list in which the inherited properties appear first, in the order listed in the superclass, followed by the properties added in this subclass.
- other properties defining the participation of the class in query operations

Also, since a Class Description object is a DMA object, it belongs to a class (`dmaClass_ClassDescription`) and has a class description property, and the DMA object contained in that property (a Class Description object for `dmaClass_ClassDescription`) will have a class description property, and so on. The apparent infinite recursion is broken by the following rule: the class description property of a Class Description object for `dmaClass_ClassDescription` is null. The following diagram presents (in simplified form) the structure of objects and metadata:

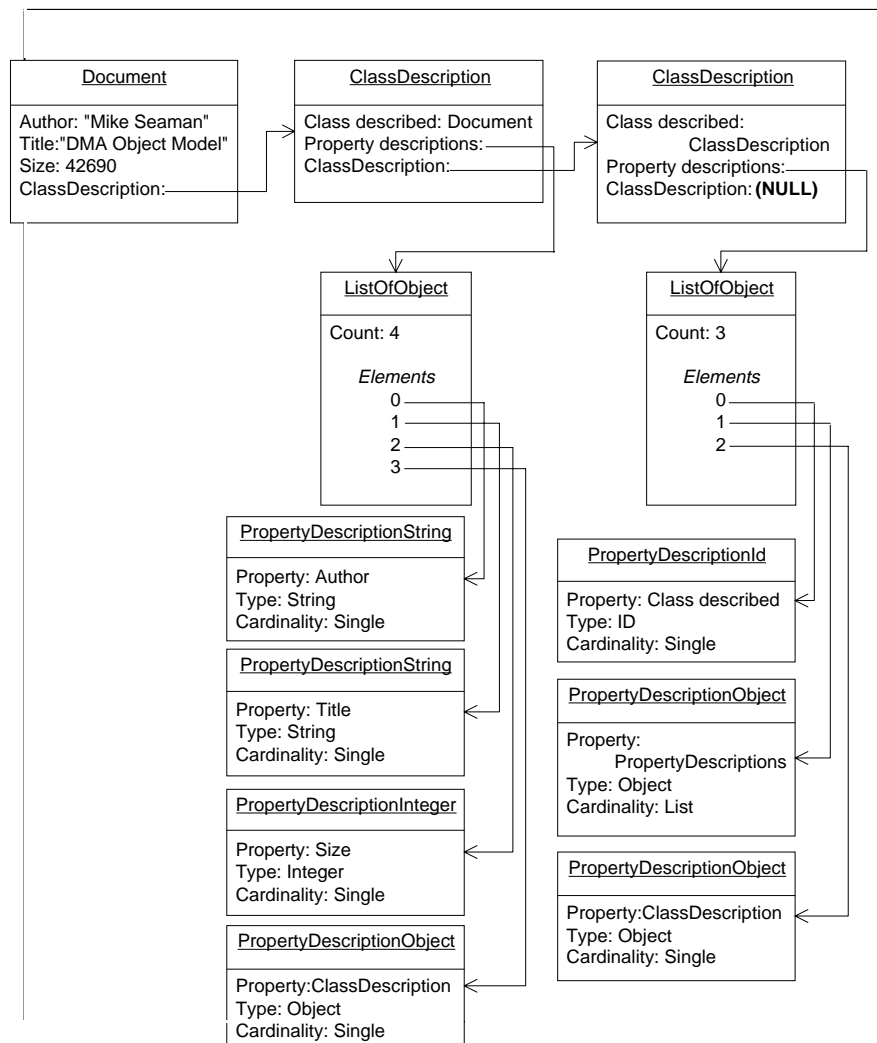


Figure 3-2: DMA Class and Property Description Classes

The diagram shows an instance of a fictional class Document having four properties: Author, Title, Size and ClassDescription. The value of the ClassDescription property is an object (instance) of class ClassDescription, having three properties: Class described, Property descriptions and ClassDescription. The ClassDescription and PropertyDescription objects shown here are only a partial representation of the actual objects.

In DMA 1.0, the class description does not specify the methods supported by objects of that class. That information is currently given only in this specification, for each well known DMA class. However, based on a priori knowledge of potential methods, COM includes an interface negotiation mechanism which allows runtime determination of the methods supported by a particular object instance.

3.1.3 DMA Properties

Properties are the principal way by which DMA objects expose (via GetProp methods) and allow manipulation of (via PutProp methods) their state. A property is a named value which may be simple or complex in nature. Properties are defined within a class and through inheritance.

3.1.3.1 Property Datatypes

A DMA property has a specified datatype. DMA allows for properties whose value is a singleton of the specified datatype (a *single-valued* property) or which contains multiple instances all of the specified datatype (a *multi-valued* property). DMA does not allow a multi-valued property consisting of heterogeneous datatypes.

The following are the permitted property datatypes:

- integer (signed 32 bit)
- string
- Boolean
- float (64 bit)
- binary
- date/time
- ID
- object

The DMA methods for accessing property values are strongly typed. There are distinct methods for accessing each property datatype. A multi-valued property must be accessed using the object datatype methods; methods are then applied to the object returned to access to the individual values of the property. The object which represents a multi-valued property can be either a list or an enumeration, both of which are described in detail below.

3.1.3.1.1 Object-valued Properties

DMA uses object-valued properties for three distinct purposes, one of which (providing access to the values of a multi-valued property) has already been mentioned. The other two purposes are:

- representing complex sub-structure of the containing object. (The analogy here is with a C data structure having a subordinate structure as one of its fields. The properties of the subobject correspond to the fields of the sub-structure. In other contexts, this would be called an *embedded* object).

- providing a means of *navigating* to a related, but independent object. (The analogy here is with a C data structure containing a pointer to another data structure. In other contexts, this would be called a *linked* object).

Object-valued properties other than enumerations (whose exceptional behavior is described later) exhibit uniform *by reference* behavior. The characteristics of this behaviour are described below:

- in a newly instantiated object all object-valued properties are in an unbound state; logically, no objects exist representing the values of those properties.
- an initial GetProp or PutProp method invocation binds the property to a specific object. In the GetProp case, this will usually be an object newly instantiated by the implementation of the GetProp method. (The only exception being the This property, for which an existing object is delivered). In the PutProp case, the invoker supplies the object.
- the binding to the object remains in effect until a further method invocation changes the binding or until the containing object is released. In the interim, GetProp calls on the property will return the same bound object every time. (Where sameness is defined per the COM specification).
- the methods which change the binding of a property are PutProp (by definition) and ExecuteRefresh. The latter restores all object-valued properties of the target to the unbound state.

This model depends upon the reference counting mechanism of COM to manage the lifetimes of objects bound to properties, hence its name.

3.1.3.1.2 Multi-valued Properties

As mentioned above, there are two types of multi-valued property: list and enumeration. Each type is represented by an object providing methods for accessing the elemental values of the property. The differences between a list and an enumeration are as follows:

- The elements of a list can be accessed in random order. An enumeration can only be traversed sequentially.
- A list may be modifiable, allowing elements to be inserted, replaced or deleted. An enumeration is always read only.
- Other than as a result of modification by the client application, a list is stable: the number, order and values of the elements remains constant so long as a reference to the list object is retained. Conversely, an enumeration is unstable in that a second sequential traversal may deliver a different number of different elements in a different order than the first traversal.

- Thus, the number of elements in a list is well defined and can be retrieved through a method call. The cardinality of an enumeration can only be determined by exhaustively traversing it and the result derived is only transiently valid.
- An enumeration of objects delivers a unique set of independently persistent objects, whereas a list of objects may deliver any kind of object, potentially with duplicates among the elements.

DMA defines list object classes (and associated methods for operating on them) for each of the base property datatypes. However, an enumerator class and methods are defined currently only for enumeration of object. DMA does not prescribe when one should be used rather than the other (except as implied by the restrictions and differences noted above), but in general enumerations are preferred where the number of elements could be larger than can be efficiently represented in list form.

Retrieval of a list object representing the value of a multi-valued property is subject to the by reference rules. In addition, for the specific case of a list of objects, each element of the list also obeys the by reference rules. That is, each element independently exhibits a not bound/bound state which results in the same object being returned on each retrieval of the element. In other words, it is as if each element were a separate single-object-valued property (of the list object).

Enumeration properties do not follow the by reference rules. An enumeration property always behaves as if it is in the unbound state. Each invocation of GetProp returns a newly instantiated enumerator object positioned at the beginning of the sequence.

A PutProp call is not allowed on a multi-valued property of either type. Where permitted, modification of a multi-valued property represented by a list is by operations on the list object, not by substitution of a different list object.

3.1.3.2 Property Naming

Like classes, properties are named by one or more globally unique identifiers. Multiple identifiers are permitted for the same reason as for class identifiers. Property identifiers are scoped relative to the class and are constrained by the rules of inheritance in the following manner: the identifiers for an inherited property must be identical in the subclass and its superclass. However, when the datatype and semantic of the property is the same, DMA allows (even encourages) the same property identifier to be used in classes having no direct inheritance relationship.

A property is accessed by specifying one of its identifiers as a parameter to a GetProp or PutProp call. In addition, since the class description specifies a well-defined ordering of the properties, alternative methods are provided which allow the intended property to be specified by its index in this ordering.

3.1.3.3 Property Metadata

The properties of a class are described by Property Description metadata objects maintained in an ordered list in a property of the Class Description object for the class.

Each property is characterized by the values of properties attached to its Property Description object, which define:

- the identifiers of the property
- the datatype of the property
- whether the property is a singleton, list or enumeration
- whether the property is optional or mandatory (i.e.: whether a value is required)
- optionally, a default value for the property
- optionally, the permitted values for the property.
- if the datatype of the property is object, the class to which the value must conform
- the participation of the property in query operations

Property description objects are classified according to the datatype of the property they describe. There is a separate class of description object for integer properties, string properties, etc. All the property description classes are derived from a common superclass (`dmaClass_PropertyDescription`) which defines the properties common to all the types of property description.

3.1.4 Metadata Spaces

A metadata space is a collection of classes related by inheritance. Concretely, a metadata space is a runtime construct materialized by an entity called the *controlling object* and represented by connected Class Description objects for the member classes. Every Class Description object is a member of a metadata space.

The defining characteristic of a metadata space is that its member classes form a single-rooted inheritance tree, and that it is closed and fully connected under class hierarchy navigation. This means that given a Class Description object for a class X and a Class Description object for class Y obtained from the `SuperclassDescription` property of `CD(X)`, one is guaranteed to find, in the `ImmediateSubclassDescriptions` property of `CD(Y)` another Class Description object for class X, `CD'(X)`, giving an identical description for X. In addition, given two classes A and B which are members of the metadata space, there exists a navigation path via `SuperclassDescription` and `ImmediateSubclass-`

Descriptions properties from a Class Description object for A to a Class Description object for B, and vice versa.

The rule of closure does not extend to other (indirect) means of navigation between Class Description objects. For example, the following navigational route may lead to an entirely different metadata space: from a Class Description to its list of Property Descriptions and from one of these (an instance of PropertyDescriptionObject) to the ClassDescription for the RequiredClass of that property. Likewise, navigation to the Class Description object of a Class Description.

There are different types of metadata space. Each metadata space contains a subset of the classes defined in this specification, and may contain other (implementation- or end user-defined) classes. Each type of metadata space is characterized by the classes that it must contain. Every metadata space must contain the class of the controlling object. Every metadata space must contain the DMA base class dmaClass_DMA as the root of the inheritance tree. Any metadata space may contain classes that fall outside its required subset, so long as the navigational closure rule is observed.

A metadata space becomes observable when the class description of the controlling object first becomes accessible, and remains observable so long as the controlling object or any instances of member classes generated from the controlling object exist. During a period of continuous observability, the metadata space is static, meaning that no classes are added or removed, no inheritance relationships change and none of the metadata for the member classes changes. This is referred to as "metadata stability".

The following types of metadata spaces can exist:

- system metadata spaces
- individual document space metadata spaces
- individual document space scope metadata spaces
- merged scope metadata spaces
- query result metadata spaces

Instances of these metadata spaces are conceptually distinct.

- The controlling object for a system metadata space is a System object. A system metadata space must contain dmaClass_System.
- The controlling object for a document space metadata space is a DocSpace object. A document space metadata space must contain dmaClass_DocSpace and all of the implemented (independently or dependently) persistable classes.

- The controlling object for a scope metadata space of either type is a Scope object of the appropriate type. A scope metadata space must contain `dmaClass_Scope`, all the searchable classes in the scope and all the classes necessary for constructing valid queries against that scope.
- The controlling object for a query result metadata space is the Query Result Set object. A query result metadata space must contain `dmaClass_QueryResultSet` and the dynamically generated result row class.

With the exception of merged scope metadata spaces, every metadata space must preserve the partial inheritance order of all of the DMA-defined classes it contains. This means that if class Q is a subclass of class P in this specification, and both P and Q are present in a metadata space then Q must be a subclass of P in that space, although with possibly more or fewer intervening levels of inheritance (including none).

In a merged scope metadata space, this rule is relaxed for the searchable classes (but only for the searchable classes). This is because a merged scope cannot in general preserve all the partial orderings of this part of the class hierarchies of the component scopes. The merged scope is permitted (but not required) to have the searchable classes appear as immediate subclasses of `dmaClass_DMA`.

The individual document space metadata and the metadata of a scope object generated from it are in conceptually distinct metadata spaces instead of conceptually being the same metadata space. The document space metadata need not include the query related classes, but the scope metadata must. Conversely, the scope metadata space need not include non-searchable persistable classes, whereas the document space metadata must include such classes. However, this conceptual distinction need not be reflected in actual practice; it is perfectly permissible for a document space and its scope to share a single runtime metadata space.

A query result metadata space is dynamically generated for each query execution. The metadata space for a query result includes only the metadata for one query and no other queries. The metadata space must include the dynamically generated result row class, which must be a subclass of `dmaClass_QueryResultRow`.

3.1.5 Persistence

Persistent objects are DMA objects which represent and provide means of modifying persistent state of the underlying document space. They operate in a fashion analogous to most editors, otherwise referred to as *scratchpad* behavior.

When such an object is first instantiated it is loaded with a snapshot of the persistent data, which forms the scratchpad. Methods are then invoked upon the

object, performing the intended changes in the state of the scratchpad object, with no immediate effect upon the persistent store. Eventually, the `ExecuteChange` method can be invoked upon the object causing it reflect the changes back to the persistent store, or the object can be discarded without saving the changes.

The snapshot which is captured in the scratchpad object offers only limited consistency guarantees with respect to the underlying persistent data. Only the single-valued non-object properties of the object are guaranteed to be populated in a consistent fashion. DMA allows multi- and object-valued properties to be evaluated lazily. Thus, the potential exists for such properties to have inconsistencies with respect to each other and to the state of the document space at the time the object was first loaded. Locking mechanisms are provided which allow a fully consistent snapshot and pessimistic protection against concurrent updates. In addition `ExecuteChange` can be induced to fail if the scratchpad object to which it is applied has been rendered inconsistent as a result of changes made from elsewhere, thereby providing optimistic concurrent protection.

At any instant, there may be many DMA objects in existence. Each such object will reflect, with varying degrees of consistency, the same underlying persistent data. There is no synchronization between these objects, even when they reside in the same process. Each is a completely independent scratchpad, and changes made via one object are not visible to another except by saving of those changes and resynchronizing the objects from the persistent store.

A mechanism is provided which allows changes to a set of objects to be made persistent in one atomic operation. This is referred to as *batching* rather than as a transaction because of the ACID properties normally expected of a transaction mechanism only the A (for atomic) is provided. (A full transaction mechanism may be defined in a future version of the DMA specification). A batch operation is performed through the `ExecuteChanges` method of a `DocSpace` object.

A persistent object may be either dependently- or independently persistent. A dependently persistent object is one which represents sub-structure of a larger independently persistent object. A dependently persistent object does not support the `ExecuteChange` method; changes to it are made persistent only when `ExecuteChange` is invoked upon the containing independently persistent object.

An object is independently persistent if and only if it has an OIID (i.e., the OIID property is supported and has a value).

An OIID is a globally unique reference to the persisted state of an object. Through mechanisms described in the Integration section, an OIID can be used to obtain a scratchpad object bound to that persisted state. The OIID is in a sense an address for the independently persistent object and so we sometimes refer to such objects as being addressable.

An object that supports the `IdmaConnection` interface is an independently persistent object. However, an instance of an independently persistent object is not required to support the `IdmaConnection` interface (for example, the instance might be read-only).

An independently persistent object supports the Create Pending, Delete Pending, and Update Pending properties.

A dependently persistent object, by definition, does not have an OIID and therefore cannot be addressed directly. A dependently persistent object can be obtained only from a property of the independently persistent object which contains it (or by a method call on that object). The snapshot principle also applies to dependently persistent object. The object must deliver a consistent snapshot of its single-valued, non-object properties, with the snapshot being taken no later than the time at which the dependent object is bound (per the by reference rules), but can lazily evaluate its multi- and object-valued properties.

A similar principle applies to a list property of a dependent or independently persistent object. A list behaves much like an object having a set of properties all of the same datatype. A non-object list must deliver a consistent snapshot of the list elements, evaluated no later than the time the list object is bound. A list of objects must consistently snapshot sufficient information to be able to deliver the element count and the element objects, although it is not required (per the by reference rules for list of object elements) to snapshot the element objects themselves until they are bound. This permissiveness with respect to the element objects implies that an object may no longer exist in the persistent store at the time an attempt is made to bind it, and therefore it is permitted for the element retrieval to fail.

3.1.5.1 Methods with Conflicting Intent

As previously described, a scratchpad object accumulates changes that are to be made later to the persistent store. It is possible for the intended operations being accumulated to logically conflict with each other. Any such conflict is resolved as soon as the conflicting method is called as opposed to when the attempt is made to apply the changes to the persistent store.

The methods that prepare the object for subsequent changes to the persistent store are grouped into two categories: primary intentional methods and secondary intentional methods.

Generally, the primary intentional methods logically conflict with each other, and so are mutually exclusive. It is an error to call two different primary intentional methods on the same object before making its changes to the persistent store. The second primary method called returns `DMARC_CONFLICTING_OPERATION`.

The same primary intentional method may or may not be allowed to be called multiple times before making the changes to the persistent store. What hap-

pens when a specific primary intentional method is called multiple times without making the changes to the persistent store between calls is detailed in the description of that method.

The secondary intentional methods do not conflict with the primary intentional methods or with each other.

The primary intentional methods are:

- `IdmaConnection:: SetDeletePending` when `DMA_TRUE` is passed as the argument
- `IdmaVersionSeries:: SetRevoke`
- `IdmaVersionSeries:: SetReserveNext`
- `IdmaVersionSeries:: SetCheckOutNext`
- `IdmaVersionable:: SetCheckIn`
- `IdmaRelationshipOrdering:: SetOrdering`

It should be noted that `SetDeletePending` can be called with `DMA_TRUE` or `DMA_FALSE` as a parameter. The value of the Delete Pending property reflects the value passed to the last `SetDeletePending` call. When the value of Delete Pending is `DMA_TRUE`, the object will be deleted when the object's changes are made to the persistent store. When an object is in this state, it conflicts with the other primary intentional methods, and calls on those methods will return `DMARC_CONFLICTING_OPERATION`. When the value of Delete Pending is `DMA_FALSE`, there is no conflict between `SetDeletePending` and the other primary intentional methods.

The secondary intentional methods include:

- `IdmaEditProperties:: PutPropVal{data type}ById`
- `IdmaEditProperties:: PutPropVal{data type}ByIndex`
- `IdmaContentTransfer:: SetCapture*`

3.1.5.2 Locking Model

Locking is a mechanism by which a consistent snapshot of an independently persistent object and its subordinate dependents can be guaranteed.

Locks are owned by DMA COM scratchpad objects, and are placed on their associated persistent object stored in the Document Space. A scratchpad object can own at most one lock on its associated persistent object. Locks are implemented by a Document Space for the persistent objects that are in the custody of that Document Space. A Document Space grants new locks on a persistent object based on the locks that are already placed on that object.

The set of methods that a client can perform on a persistent object is constrained by the locks that exist on that object, if any. A lock is considered to be a resource owned by the scratchpad object and should not persist beyond the lifetime of that object. When a document space connection is released or broken, any existing locks that were placed on scratchpad objects obtained via that connection are removed.

Only existing independently persistent objects can be locked. The methods that implement the DMA locking model for persistent objects are ApplyLock, ConnectAndLockObject and RemoveLock. An object that has not yet been made persistent can not be locked until after it has been made persistent. Purely synthetic objects can not be locked.

It is not required to have a lock on a persistent object before modifying its properties or deleting it. However, if there is a lock on a persistent object, and an attempt is made to modify or delete that object, then the modification or delete operation may be disallowed, depending on the type of lock and on which DMA COM object owns the lock.

The locking methods are synchronous, and the operation is resolved before they return to the caller. The ApplyLock and ConnectAndLockObject methods do not block. The ExecuteChange and ExecuteChanges methods have no effect on locks on persistent objects. Thus, these two methods do not cause the placement or removal of locks on persistent objects, with the exception that if they delete a persistent object, any locks placed on it are deleted as well. (Because read locks and existence locks prohibit deletion of the object, and because a write lock precludes other write locks, the deletion will remove the sole write lock on the object (if one exists and the scratchpad object being is used is the one that owns the write lock))

One consequence of the locking model is that locking errors can be returned by a number of different methods, including ExecuteChange and ExecuteChanges.

There are three types of locks: (1) read lock, (2) write lock, and (3) existence lock.

A **read lock** is compatible with read locks and existence locks, but is not compatible with write locks. Multiple read locks can be placed on a persistent object simultaneously. A write lock cannot be placed on a persistent object that already has a read lock on it. If there is a read lock on a persistent object, that object may not be modified or deleted via any connection, but it may be retrieved via any connection.

An **existence lock** is compatible with other existence locks, read locks, and write locks. Multiple existence locks can be placed on an object simultaneously, along with multiple read locks, or with a single write lock. A write lock can be placed on an object with an existence lock on it. If there is an existence lock on

a persistent object, that object may not be deleted via any connection, but it may be modified and retrieved via any connection.

A **write lock** precludes other write locks and read locks. At most one write lock can be placed on a persistent object, and while it is in effect, no other read or write locks can be placed on that persistent object. If there is a write lock on an object, the object can be modified, or the object can be deleted if there is no existence lock on the object, but only via the scratchpad object that placed the write lock. However, a persistent object can still be retrieved via any document space connection, even if there is a write lock on it.

Any persistent object may be connected to and retrieved, or its properties returned as part of a query, via any document space connection, regardless of the existence of locks on the object or the kinds of those locks.

A distinction must be made between a DMA COM scratchpad object and the underlying persistent object with which it is associated. Some of the properties of a DMA COM object may be synthetic object valued properties. For example, a canned query may be executed in order to instantiate a DMA COM object for the value of such a property. In this case, a change in the value of the synthetic property can result as a side effect of a change to a separate independently persistent object. Locking a persistent object will not prevent this type of change to the values of the synthetic properties of any DMA COM objects with which it is associated.

For example, a document space may implement direct containment such that there is no independently persistable direct containment relationship object – it is purely synthetic. Therefore, the object to be contained has a property that is modified when the contained object is inserted or deleted from a direct container. Locking the object to be contained, therefore, can prevent insertion or deletion of that object into a direct container in such an implementation.

As another example, referential (or direct) containment may be implemented by a particular document space such that neither the object to be contained nor the container object have any properties that are modified when the object to be contained is inserted or deleted from the container. An independently persistable relationship object is the only object created, deleted, or modified in the document space database. In such an implementation, the locking model does not prevent insertions or deletions into referential (or direct) containers.

Similarly, in the case of versioning, if any of the objects involved (i.e., Configuration History, Version Series, Version Description, or DocVersion) are independently persistable (as opposed to being synthetic objects), the locking model can prevent checkout, checkin, etc. by locking the independently persistable objects whose properties are changed as a result of such methods.

3.1.6 Objects and Implementations

The separation of interface from implementation referred to earlier allows the same DMA class of object to be implemented in different ways by different vendors while exhibiting the same behavior (with certain permitted variations described in the Extensibility section below). This allows an application to operate on objects using only the mechanisms defined in this specification, and assures it of being able to accomplish its task regardless of who supplied the implementation.

This principle extends to objects that operate to other objects. When one DMA object operates on another, it must, **in general**, be able to accomplish its task using only the mechanisms defined in the DMA specification, without any reliance on implementation-specifics. In other words, DMA intends the public interfaces and properties defined in this specification to be adequate for implementing the interoperability guarantees of this specification. Those interoperability guarantees include searching across repositories and the copying of content from one document space into another.

Where this general rule applies, the operand object may be implemented in any fashion consistent with the specification for the class to which the object belongs; in particular the object may be implemented by the client application itself. However, there may be performance advantages in instantiating such objects from the document space or scope to which it is to be delivered, because implementations are free to have optional hidden interfaces to accelerate performance.

However, there are a number of specific cases where this rule does not apply, because the genesis of the operand object (and hence its implementation) is mandated by this specification. In such cases, any attempt to use an object generated in some other way will result in a runtime error. The objects to which this latter mandate applies are the following:

- the System Manager object, which may only be generated by a call to the DMA “bootstrap” function `dmaConnectSystemManager`.
- a System object, which may only be generated by a call to the `ConnectSystem` method of the System Manager object.
- a DocSpace object, which may only be generated by a call to the `ConnectDocSpace` method of a System object.
- a document space Scope object, which may only be generated by a call to the `GetScope` method of a DocSpace object.
- independently persistent objects, which may only be generated directly or indirectly from a DocSpace object representing the document space in which the object is, or is to be persisted. Direct generation would be by either the `CreateObject` or `ConnectObject` methods of the DocSpace object.

Indirect generation occurs through a variety of different methods on objects which were themselves directly or indirectly generated from the DocSpace object. In particular, the `CreateInstance` method of the Class Description object for the intended class can be used to generate appropriate new objects so long as the Class Description object itself was generated (directly or indirectly) from the DocSpace object.

Note that dependently persistent objects are not the subject of any mandate and so follow the general rule. However, a dependently persistent object which is not delivered (via `CreateObject` or `CreateInstance`) by the DocSpace in which it is to be made persistent, must in any case be constructed in such a way to conform to the class description of that document space.

Object implementations may cooperate in establishing private mechanisms by which they offer additional functionality or enhanced performance. The only case in which an implementation may depend on those private extensions being available on another object is when the implementation for the object is known through mandate. In such cases, any appropriate implementation technique may be used to access in the private extensions. In all other cases, the private extensions must be represented as additional COM interfaces on the object such that their presence or absence can be determined at runtime through COM interface negotiation.

3.1.7 Required and Optional Features

The DMA class hierarchy defined in this specification and the properties specified for those classes are complete for the purpose of providing all the required and optional features currently specified by DMA. However, this does not mean that all these classes and properties must be present in the runtime metadata of every implementation. Indeed, DMA requires that classes and properties which represent optional features that are not implemented be absent from the runtime metadata. This allows a DMA application to determine the presence or absence of those features from the metadata.

The minimum set of classes which must be implemented in order to meet the baseline requirements for a DMA service is specified in the Conformance section, as are the additional classes which must be provided for each optional feature. In general, if circumstances require a particular class to be present in the runtime metadata, then its superclasses (up to and including `dmaClass_DMA`) must also be present. However, there are two specific cases where this rule does not apply, because the superclasses are representative of optional features:

- If no form of containment is supported, then `dmaClass_Containable` must be omitted from the runtime metadata, and any implemented classes which would otherwise appear as immediate subclasses of `Containable` (for example, `dmaClass_Versionable`) will instead appear as immediate

subclasses of `dmaClass_DMA`. In addition, `dmaClass_Container` and the containment relationship classes must be omitted.

- If versioning is not supported, then `dmaClass_Versionable` must be omitted from the runtime metadata, and any implemented classes which would otherwise appear as immediate subclasses of `Versionable` will instead appear as immediate subclasses of `dmaClass_Containable`. (Or `dmaClass_DMA` if containment is also not supported). In addition, `dmaClass_ConfigurationHistory`, `dmaClass_VersionSeries`, `dmaClass_VersionDescription` and `dmaClass_Reservation` must be omitted.

Note however that if containment (or versioning) is supported for some classes that are specified to be subclasses of `Containable` (or `Versionable`), then class `Containable` (or `Versionable`) must be present and the partial ordering rule must be observed. This may result in some classes which are not containable (or versionable) appearing as subclasses of `Containable` (or `Versionable`). Therefore, determination of whether a particular object is containable (or versionable) must be done by examination of capabilities or property metadata. An `IsOfClass` test will not suffice.

If circumstances require a class to be omitted from the runtime metadata, then its superclasses up to but **not** including `dmaClass_DMA` should also be omitted, unless their presence is required for some other reason. (I.e. that some other subclass is implemented). For example, if containment is not supported and therefore `dmaClass_DirectContainmentRelationship` and `dmaClass_ReferentialContainmentRelationship` are omitted, then `dmaClass_ContainmentRelationship` and `dmaClass_Relationship` may also be omitted.

The Object Reference section specifies the properties which must be implemented (i.e. must be present in the runtime metadata) for each class, *assuming the class itself is implemented*. If a property is not required, then it is representative of an optional feature, and should be present or absent according to whether or not the feature is supported *for that class*. For example:

- The `OIID`, `Create Pending`, `Delete Pending` and `Update Pending` properties should only be present for classes which are independently persistable.
- If a particular class is not capable of being referentially contained, then the `Containers` property should be omitted for that class.

Note that implementation of an inherited optional property in a particular class does not imply that the property must be present in the metadata for the superclass in which the DMA specification introduces that property. That would imply, for instance, that the `OIID` property would be required to be present in the runtime metadata for `dmaClass_DMA` (and hence all classes), completely defeating the purpose of it being optional.

3.1.8 Extensibility

DMA provides several avenues by which a document space vendor can provide variations to and extensions of the behavior defined by this specification.

- First, of the classes and properties defined by this specification, only certain characteristics are mandated, with the remainder left to the discretion of the document space. This includes the option to omit certain features and/or properties as well as allowing some variability in the classes and properties which are supported.
- Second, a document space may add properties to any of the DMA-defined classes without changing the class identifier. This option is also open to the DMA in future versions of this specification.
- Third, objects may be offered with support for additional methods not defined by DMA. This option is of lesser utility insofar as interoperability is concerned since DMA objects are not self-describing with respect to methods.
- Fourth, any of the DMA-defined classes may be subclassed. This is mostly likely to be done at the leaves of the inheritance tree and in particular to the DocVersion class, but is not restricted to this. Included in this option is the ability to introduce new intermediate classes into the DMA inheritance hierarchy, so long as the partial ordering of existing classes is preserved. Subclassing the DMA leaf property classes is the most highly preferred method of extending DMA.
- Fifth and last, a document space may offer additional query operators and collating sequences beyond the well known sets defined in this specification.

A document space vendor may also offer extensibility in any of the forms described above as an end user customization feature. Mechanisms for such customization are not defined by the DMA 1.0 specification. Adding properties and classes is preferred over adding interfaces and methods. Additional well known query operators, well known collating sequences, and well known properties may be added to the DMA specification in the future.

3.1.9 Evolution of Persistent Metadata

The metadata in the persistent store of the document space may evolve over time. The metadata can be changed off line (i.e., when no DMA clients have valid connections to the document space), and, some systems may also allow it to be changed on line (i.e., when there is at least one DMA client with a valid connection to the document space).

When the persistent metadata is changed off line, the expected behavior will occur after the change. If the change is upward compatible, DMA clients will

continue to operate as before the change. If the change is not upward compatible, some problems may occur in DMA clients that were previously operating correctly.

When the persistent metadata is changed on line, then DMA clients that are active at the time may subsequently receive the error code `DMARC_STALE_METADATA` for certain methods. That is because the client's metadata is stable, and so does not match (to some degree) the modified metadata in the persistent store.

The key benefit of metadata stability is that the client can count on information derived from the metadata remaining usable without reconsulting the metadata. A DMA application is free to cache information and to carry derived information in its operations with the assurance that the metadata space will not have changed.

The metadata mismatch may or may not matter when a particular query or update is attempted on the persistent store of the document space. If the metadata mismatch doesn't matter for the operation at hand, the document space implementation may choose to attempt to complete operation and to not return the `DMARC_STALE_METADATA` error code, or it can return the error without attempting to see if the operation can proceed. If the mismatch does matter, the `DMARC_STALE_METADATA` error code must be returned.

The reference sections of this specification define which methods are permitted to return `DMARC_STALE_METADATA`. The general principle is that only methods which bind or modify a persistent object may return the error. Methods which navigate the client's (stable) copy of the metadata or operate on existing scratchpad objects may not return the error.

3.1.10 Document Space Connections

The document space object has a logical connection to the persistent store of the document space. In a distributed system, this connection is implemented on top of a conceptual network connection between the host on which the DMA client runs, and the host on which the persistent store is resident. Some document spaces will require that the client be authenticated before the logical connection is fully operative. Objects created from the document space (e.g., an object that is a subclass of `DocVersion`) exploit the logical connection of the document space object (for example, by calling `IdmaConnection::ExecuteChange`).

The logical connection between a document space object and the persistent store can be broken by calling `IdmaDocSpace::ExecuteDisconnect`. Once the logical connection to the persistent store is broken, methods on objects that need that logical connection (e.g., `ExecuteChange`) return `DMARC_DISCONNECTED` errors permanently.

The document space object's conceptual network connection can be interrupted, give errors, or be broken. In such a case, DMARC_NETWORK_ERROR or DMARC_LOST_CONNECTION might be returned. In addition, platform specific error codes might be returned. The DMARC_NETWORK_ERROR error code gives no information as to whether a subsequent retry might succeed. In contrast, the DMARC_LOST_CONNECTION error code guarantees that the document space connection has become permanently unavailable.

When initially attempting to make a network connection, DMARC_NETWORK_UNAVAILABLE may be returned.

3.2 DMA Interface and Process Model

3.2.1 DMA Interface Model

The interface mechanism of DMA is a strict, portable subset of Microsoft's COM (Component Object Model). You can use the Microsoft publication "The Component Object Model Specification", Version 0.9, October 24, 1995 (or any later version) as a reference (<http://www.microsoft.com/com>).

The DMA Architecture uses only the basic encapsulation mechanism of COM for local, memory resident objects. (This is sometimes referred to as the "binary interface standard".) This mechanism gives the DMA API a language independent, binary component architecture. The binary interface standard is a tractable subset of COM, and it is relatively easy to understand and implement. The DMA Architecture has no dependency on DCOM, IDL, MIDL, ODL, DCE, or any other remote procedure call mechanism or wire protocol. The network aspect is left open, and is transparent to the DMA API. The DMA Architecture has no dependency on any aspect of OLE, including persistent storage.

3.2.1.1 COM Elements Used By DMA

3.2.1.1.1 GUID

The DMA Architecture and COM use GUID's (Globally Unique IDentifiers) to uniquely identify COM interfaces. These GUID's are 16 byte quantities. The 128 bit pattern that a GUID represents is virtually guaranteed to be unique over all space and time, but no central administration authority is required to achieve this uniqueness. In other words, every GUID ever generated by any computer is different from every other GUID generated by that computer or any other computer. COM's GUID's are actually DCE UUID's (Universally Unique IDentifiers). Multiple algorithms for generating GUID's exist, and these algorithms are publicly available. There are utilities and entry points available on Microsoft and UNIX platforms to generate GUID's. A GUID is represented as a C typedef struct, and can be statically initialized in program header files.

Note: The DMA API data type `Dmald` is a GUID. The DMA API also uses `Dmald`'s to uniquely identify things that have nothing to do with COM. For example, the DMA API uses GUID's to uniquely identify properties, property classes, query operators, collating sequences, etc.

3.2.1.1.2 Object and Interface Model

The DMA Architecture uses the basic object and interface model of COM. COM object instances have internal state in the local memory of the process and support multiple interfaces. Each interface contains only signatures of pointers to functions and supporting type declarations -- no variables. All DMA

interfaces are COM interfaces. Each COM interface is assigned a unique, invariant GUID.

A COM interface definition maps well to a C++ pure virtual class. A COM interface can also be described in C or C++ by declaring a typedef for a pointer to a C struct that has fields which are pointers to functions.

A COM interface contract is the interface definition plus documentation describing the semantics of the methods. A COM interface contract is specified only once, is independent of object type, and cannot be changed once it is published. Many different types of objects can implement and provide the contract of a COM interface.

The COM specification provides for single inheritance of COM interfaces. That is, a COM interface can inherit the contracts of other COM interfaces in a well defined, ordered linear sequence. However, COM does not provide for inheriting the implementation of interfaces, only the definitions and contracts of the interfaces. A COM interface that inherits another COM interface inherits the entire contract of that interface, and must provide all of the functionality of the inherited contract to its clients. There is no selective inheritance of only some of the methods of the inherited interface, and it is not permissible to change the contract of the interface being inherited.

If a COM interface B inherits COM interface A, then A comes before B in the interface inheritance ordering, and all the methods of A plus all the methods of B are provided by interface B. The interface being inherited (in this example, interface A) is called a “base interface”.

COM single-interface-inheritance must be distinguished from several other types of inheritance:

- Interface inheritance, i.e., C++ class inheritance, is a C++ language concept that has no counterpart in the C language. Single interface inheritance of pure virtual C++ classes happens to map well to COM single-interface-inheritance. Conceptually, however, C++ interface inheritance is distinct from COM single-interface-inheritance. For example, in C++ one could choose to specify COM interfaces either as pure virtual C++ classes, or as a pointer to a C struct containing pointers to all the methods of the interfaces. And in C++ one can perform single inheritance or multiple inheritance of C++ classes, and can use interface inheritance for classes other than pure virtual classes.
- Implementation inheritance is provided by the C++ language, but it is not provided by COM.
- The DMA single-inheritance class hierarchy defined in this specification (see Object Reference) is distinct from COM interface inheritance, C++ class inheritance, and implementation inheritance.

3.2.1.1.3 IUnknown Interface

The IUnknown interface was defined by Microsoft, and, as per the COM specification, is the ultimate base interface for all COM interfaces. Thus, the IUnknown interface is first in the COM interface inheritance ordering of every COM interface, and the methods of the IUnknown interface are available in every COM interface.

Hence, all DMA objects support the IUnknown interface, and the methods of the IUnknown interface are available in every DMA interface, as required by the COM specification.

The QueryInterface method of the IUnknown interface enables a client of a COM object (and, therefore of a DMA object) to obtain access to any interface of the object of which the client has knowledge. The methods of the IUnknown interface are also used in controlling the lifetime of the object according to the COM reference counting rules as per the COM specification.

3.2.1.1.4 Memory Management Rules

The DMA Architecture adopts the memory management rules for reference counting on COM objects (to manage their life cycle), and for the ownership responsibilities of input, output, and in/out parameters to methods.

DMA employs the standard IMalloc interface to allocate local task memory dynamically. The DMA System Manager provides a default allocator. However, the DMA client has the ability to override the default allocator and substitute one of its own choosing.

3.2.1.1.5 Error Handling

The DMA Architecture follows COM in disallowing setjmp/longjmp and all exception throw/catch mechanisms across a COM interface boundary. Such mechanisms conflict with transparent remoting of DMA interfaces and are compiler and language dependent.

3.2.1.1.6 Error Result Codes

DMA follows the COM convention that virtually all methods return a 32 bit integer value called the result code, which indicates the completion status of the method call. These values conform to the COM HRESULT scheme as described in <http://www.microsoft.com/oledev/olecom/Ch03.htm#Error>. DMA however does not employ the HRESULT typedef in any of its header files. Instead DMA uses the type DmaRC, which should be treated as a synonym for HRESULT.

Two types of result codes exist:

- 1 Error codes, and
- 2 all other result codes.

Error codes indicate that the method failed completely and cannot deliver any useful results. The COM convention is that a method which returns an error code must return NULL values for any output pointer parameters.

All non error result codes indicate some degree of successful operation, with anything other than unqualified success (DMARC_OK) being considered a warning of incomplete success. When a warning code is returned, output parameters should deliver useful values which allow the application to determine the extent to which the operation was incomplete.

An error result code is distinguished from a warning or success code by the fact that the most significant bit in the value will be set. The DMA header files provide convenience macros to test for an error result code.

The HRESULT scheme separates result codes into a number of different categories, identified by *facility* codes that are encoded into the high order part of the HRESULT value. Microsoft reserves the sole right to define facility codes and to define the result codes belonging to every facility except one: FACILITY_ITF. Anyone may define result codes belonging to FACILITY_ITF. However, no mechanism exists to prevent two or more definitions using the same value for different purposes. It is therefore necessary that result codes belonging to FACILITY_ITF only be interpreted relative to the particular interface which returned the code, and that FACILITY_ITF result codes not be propagated across boundaries between interface families. DMA defines result codes in FACILITY_ITF for the DMA family of interfaces.

The COM convention is that any result code, in any facility, may be returned by any COM-defined interface method. COM-defined methods usually include, as part of their specification, a list of "particularly useful" error codes, but, in accordance with the HRESULT scheme, the list is never exhaustive. DMA honors the COM conventions for all result codes, except for those belonging to FACILITY_ITF. For FACILITY_ITF result codes, DMA imposes the following additional rules:

- The only FACILITY_ITF result codes that may be returned by a DMA-defined method are those given in the reference section of this document (see Return Codes) and identified in the DMA header files by symbolic constants with names of the form DMARC_<name>.
- The specification of every DMA method lists the FACILITY_ITF result codes that the method may return. Although the method may return any result code belonging to a facility other than FACILITY_ITF, the method

may only return a FACILITY_ITF result code if it is specifically listed for that method, or if it is one of the global DMA error codes mentioned below.

It should be noted that the preceding two rules are imposed upon DMA **implementations**, and that the list of permitted result codes may be added to in subsequent revisions of the DMA specification. These rules give no guarantee to the DMA **application** as to what result codes can be expected (since the application may encounter an implementation which was constructed against a later version of the specification than it was). Therefore, DMA compliant applications must adhere to rules given in the COM specification with regard to treatment of unrecognized result codes.

The DMA header files do not reproduce any definitions for any of the standard COM result codes. (For example, there is no symbolic constant definition for E_NOINTERFACE.) However, for a few of the standard COM result codes, especially those which are required by the IUnknown interface, an alias is defined (in the form of a "DMARC_" symbolic constant). This is indicated in the description of the result codes given in the result codes reference section. When such result codes are listed as a possible return of a DMA method, the listing is done in the same spirit as in the COM specification.

Some methods list two sets of permitted result codes, with the second set described as "Deferred". A deferred result code is one which may be returned by that method and may also be returned by the ExecuteChange or ExecuteChanges calls which subsequently enacts the effect of the method on the persistent store. Such result codes are not also listed for ExecuteChange or ExecuteChanges.

A number of DMA error codes are sufficiently general that DMA permits them to be returned by practically any DMA method. The specification does not systematically list these for every method. They should be presumed as possible result codes for every method. The "global" error codes are as follows:

- DMARC_DISCONNECTED
- DMARC_LOST_CONNECTION
- DMARC_NETWORK_ERROR

3.2.1.1.7 Transparent Remote Access

The DMA Architecture follows COM in disallowing anything that would stand in the way of making method calls transparently remoteable.

The DMA Architecture is silent on the network RPC and protocol technologies used, and so allows a variety of technologies.

3.2.1.2 COM Elements Not Used By DMA

COM elements not used by the DMA Architecture include the following:

- Persistent Storage Interfaces
- Monikers
- Uniform Data Transfer
- Microsoft OLE
- Object Broker or any other Object Request Broker
- COM Client/Server Protocol
- COM Aggregation
- IDL (Interface Description Language), MIDL
- COM library, including CoInitialize(), CoUninitialize(), and CoCreateInstance().
- COM registry
- IClassFactory Interface
- Server Management
- COM servers
- COM library multithread services

3.2.1.3 Additional Requirements on DMA Objects

3.2.1.3.1 Property Classes

All DMA objects have exactly one associated DMA property class. The object's (DMA property) class is used to describe the type of DMA object. Each type of DMA object has a set of interfaces that it must support, and a set of interfaces (which may be empty) that it may optionally support.

3.2.1.3.2 Required interfaces

In addition to the IUnknown interface (which must be supported by all COM objects), DMA requires all DMA objects to support the IdmaProperties and IdmaObject interfaces. The IdmaProperties interface provides access to the properties of the DMA object.

3.2.1.3.3 Object Generation

The first DMA object is obtained by calling an entry point procedure in a dynamically linked library. From then on, all DMA objects are obtained by invoking a method on a DMA object that is already in hand.

3.2.2 DMA Process Model

3.2.2.1 Objects Are Local to the Process

The object instance state is local to the process. On some platforms, the procedure pointers to the methods are only valid for the current process. DMA objects allocated in one process cannot be accessed by a different process.

On platforms that support threads, all threads have access to the local memory of the process, so all threads of a process can share access to the DMA objects of the process.

3.2.2.2 Thread Safety

DMA requires that object instances be safe under unrestricted threading (wherein multiple threads of a process can run both user and OS code concurrently). This requirement means that all methods of all interfaces of DMA objects must perform locking adequate to prevent destructive interference when multiple threads call methods on the same object concurrently. It is sufficient that DMA objects be serially reusable.

Serial reusability can be accomplished by single threading all access to the object via its methods. However, in one or two instances, it is desirable, but not required, to allow a second thread of the current process to call a method while the first thread is in the middle of executing another method on the same object. For example, it is desirable that `TerminateResults` be callable by a second thread while the first thread is blocked in `GetNextResultRow` during retrieval of query results. This type of implementation would allow long running queries to be canceled by a second thread responsive to user input while the first thread is blocked waiting for more result rows.

3.2.2.3 Callbacks

Certain potentially long running methods take an optional callback parameter. The method is allowed, but not required, to call the callback procedure at intervals during the operation. Callback procedures, when supported, may be used to facilitate the client's reporting on the progress of the operation or to cancel the operation.

3.3 DMA Integration Model

3.3.1 Introduction

The DMA Integration Model defines how DMA components are assembled, how these components interact and how client applications interact with these components.

The principal DMA components visible to DMA client applications are as follows:

- System Manager Object
- System Objects
- DocSpace Objects

Additional components participating in the integration infrastructure are follows:

- Text Ordering Service Objects
- OIID Parser Service Objects
- Scope Factory Service Objects

The System Manager is a COM object that can create new DMA System objects, which in turn can create new DMA DocSpace objects.

DMA Text Ordering service objects enable text collation logic to be decoupled from DMA System or DocSpace object implementations.

DMA OIID Parser service objects enable OIID parsing logic to be decoupled from the DMA System Manager implementation.

DMA Scope Factory service objects enable merged scope creation to be decoupled from the DMA System object implementation.

The System Manager is implemented in an operating environment dependent manner. AIIM intends to provide System Manager implementations for popular environments including Microsoft Windows 32 and various Unix platforms. The mechanism that clients utilize to create System Manager objects is both platform and language specific.

DMA *service providers* implement the remaining objects. A uniform (although platform specific) mechanism is defined by which service providers can be called upon to deliver instances of the objects they implement.

3.3.1.1 Integration Model Principles

The following UML figure illustrates the relationships between the DMA Integration model components.

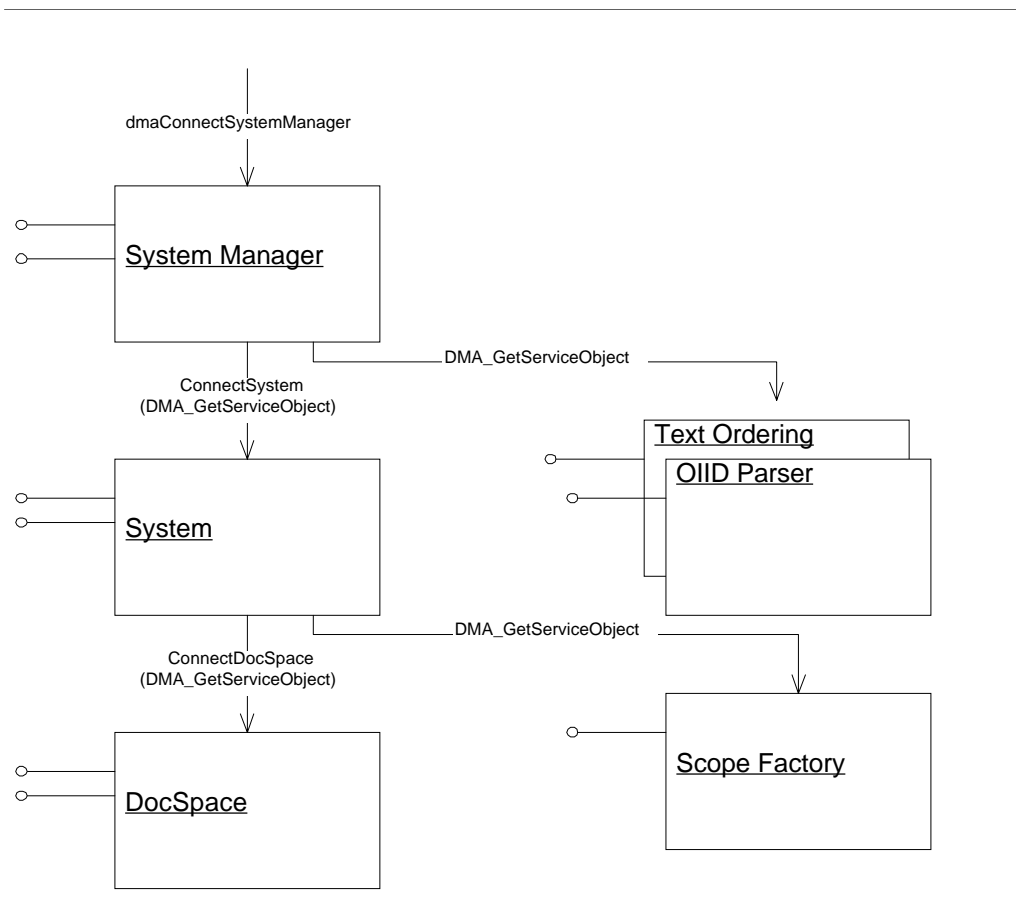


Figure 3-3: DMA Integration Model Components

DMA doesn't specify a traditional functional client API. Instead, the DMA client API can be viewed as one function which delivers System Manager objects and a set of the methods in COM interfaces delivered by those System Manager objects which in turn deliver other objects.

Similarly, DMA doesn't specify a traditional functional server side SPI. Instead, DMA specifies a set of COM interfaces that DMA Service Objects must implement, COM interfaces for registering DMA Service Objects and a single function for activating a DMA Service Object.

DMA also doesn't impose restrictions on the distribution of DMA implementation other than requiring that the initial means of obtaining the initial System Manager object be callable by the client application. As the rest of the client's interaction is via method calls on COM interfaces, the underlying implementation of any of the DMA COM objects may be remote from the client's point of presence.

3.3.2 The Client View: Access to DMA Systems and Document Spaces

3.3.2.1 Application Connection to the System Manager

A DMA application obtains (connects to) a System Manager object by a well known mechanism. The only currently defined mechanism supports C and C++ client implementations on the Win32 platform. Virtually identical mechanisms are likely to be specified for C and C++ applications on other platforms. This mechanism may or may not be accessible from other implementation languages; where it is not, DMA may define alternative mechanisms in the future.

For DMA 1.0 on the Win32 platform, a System Manager object is obtained by calling the `dmaConnectSystemManager` function, which is exported by the `DMA10.DLL` dynamic link library. On other platforms, the same function will be used, although the packaging of that function (e.g. shared library, static library, etc.) is not currently defined.

The following is the general signature for `dmaConnectSystemManager`.

```
DmaRC dmaConnectSystemManager (
    pDmaVoid pEnvInfo,
    pDmaOpaqueString pProfile,
    Dmapv pIMalloc,
    DmaInteger32 lCharSetEncodingID,
    pDmaString pLocaleName,
    REFIID riid,
    pDmapv ppISysMgr)
```

The following sections discuss how an application selects values for each of these parameters.

The *pEnvInfo* parameter should be NULL or operating environment specific data that is required to define the execution environment that the System Manager object must operate within. It's required value will be specified for each operating environment System Manager implementation. For Win32, this parameter should be NULL.

The *pProfile* parameter should be NULL or operating environment specific data that is required to locate System Manager configuration information such as a registry key name. Allowable values will be specified for each operating environment System Manager implementation. For Win32, this parameter should be NULL.

The *pIMalloc* parameter should be NULL or a reference to a COM interface of an object that supplies an IMalloc interface. If this parameter is NULL, the System Manager provides a default IMalloc implementation that is used throughout the lifetime of the System Manager and any of its children. For the Win32 platform, the default implementation is that returned by the `CoGetMalloc` API.

The System Manager and all of its children will employ this IMalloc interface in constructing all storage structures delivered as output parameters from methods on their interfaces. This applies any time that an interface delivers an output pointer that is not a pointer to an interface, including pointers embedded in the data reached by such outputs (hereafter referred to as *non-interface output pointers*). In addition, any interfaces supplied as parameters to methods on the System Manager or its must deliver non-interface outputs that were allocated with this same IMalloc interface.

The *ICharSetEncodingID* parameter specifies a character set encoding that the System Manager object and all its children will utilize for any DmaString parameters. Its value will typically specify Unicode when operating in a Microsoft COM environment.

The *pLocaleName* parameter specifies the name of a locale that the System Manager object and any System objects returned via the System Manager's EnumerateSystems method are requested to operate within. A NULL value indicates that the System Manager and System objects returned from EnumerateSystems may choose a default locale to operate within based upon implementation choices or operating environment information, for example the current Windows or Unix locale.

The DMA Internationalization and Localization chapter discusses character set encoding and locale names in more detail.

The *riid* parameter must be a reference to an interface identifier (IID) for an interface supported by a System Manager object. This will typically be a reference to IID_IdmaSystemManager.

The *ppISysMgr* parameter is used to return a reference to the requested interface on a System Manager object.

If the **dmaConnectSystemManager** function is successful, a DMARC_SUCCESS return code value is returned, otherwise one of a small set of error return codes will be returned.

3.3.2.2 Accessing DMA Systems

A DMA application obtains (connects to) a DMA System object using methods on the System Manager object's IdmaSystemManager interface. The EnumerateSystems method is used to return an IdmaEnumOfObject interface that can be used to enumerate all of the System objects that a System Manager can connect to. A specific System can be connected to using the ConnectSystem method.

DMA Systems are the first full fledged DMA object that an application encounters. They support the IdmaObject and IdmaProperties interfaces and as such exist within a DMA class hierarchy, have properties and are described by DMA meta-data.

The properties specific to a System object include a Display Name, a Descriptive Text, a list of supported Locale Names and the current Locale Name. The values of the Display Name and Descriptive Text properties would typically have values based upon the value of the current Locale Name property.

The System objects returned by the EnumerateSystems and ConnectSystem methods may represent unauthenticated connections to DMA systems. If the system requires user authentication, it will indicate so by providing an IdmaAuthentication interface on the System object.

Applications should determine whether authentication is required before attempting to invoke any methods other than those used to browse the object's property values and meta-data. If the interface is not present, the object does not require explicit authentication. Otherwise authentication can be achieved using methods available on the IdmaAuthentication interface. The use of these methods is described in more detail in a later section.

Object valued properties cannot be accessed if the System object is in an unauthenticated state. Attempts to access an object valued property in an unauthenticated state will yield a DMARC_NOT_AUTHENTICATED return code.

3.3.2.3 Accessing DMA Document Spaces

A DMA application obtains (connects to) a DMA DocSpace object using methods on the System object's IdmaSystem interface. The EnumerateDocSpaces method is used to return an IdmaEnumOfObject interface that can be used to enumerate all of the DocSpace objects that a System can connect to. A specific DocSpace can be connected to using the ConnectDocSpace method.

The properties specific to a DocSpace object include a Display Name, a Descriptive Text, a list of supported Locale Names and the current Locale Name. The values of the Display Name and Descriptive Text properties would typically have values based upon the value of the current Locale Name property.

The DocSpace objects returned by the EnumerateDocSpaces and ConnectDocSpace methods may represent unauthenticated connections to DMA document spaces. If the document space requires user authentication, it will indicate so by providing an IdmaAuthentication interface on the DocSpace object.

Applications should determine whether authentication is required before attempting to invoke any methods other than those used to browse the object's property values and meta-data. If the interface is not present, the object does not require explicit authentication, otherwise authentication can be achieved using methods available on the IdmaAuthentication interface. The use of these methods is described in more detail in a later section.

Object valued properties cannot be accessed if the DocSpace object is in an unauthenticated state. Attempts to access an object valued property in an unauthenticated state will yield a *not authenticated* return code.

Applications may use DMA DocSpace objects to instantiate objects managed by the document space. These objects may be instantiated from a DocSpace object using DMA addressability, navigation and searching which are discussed in the DMA Object Model and DMA Query Model sections of this document.

3.3.2.4 Client Access Summary

The following figure and notes describe how character set encoding and locale name data affect the DMA client integration model.

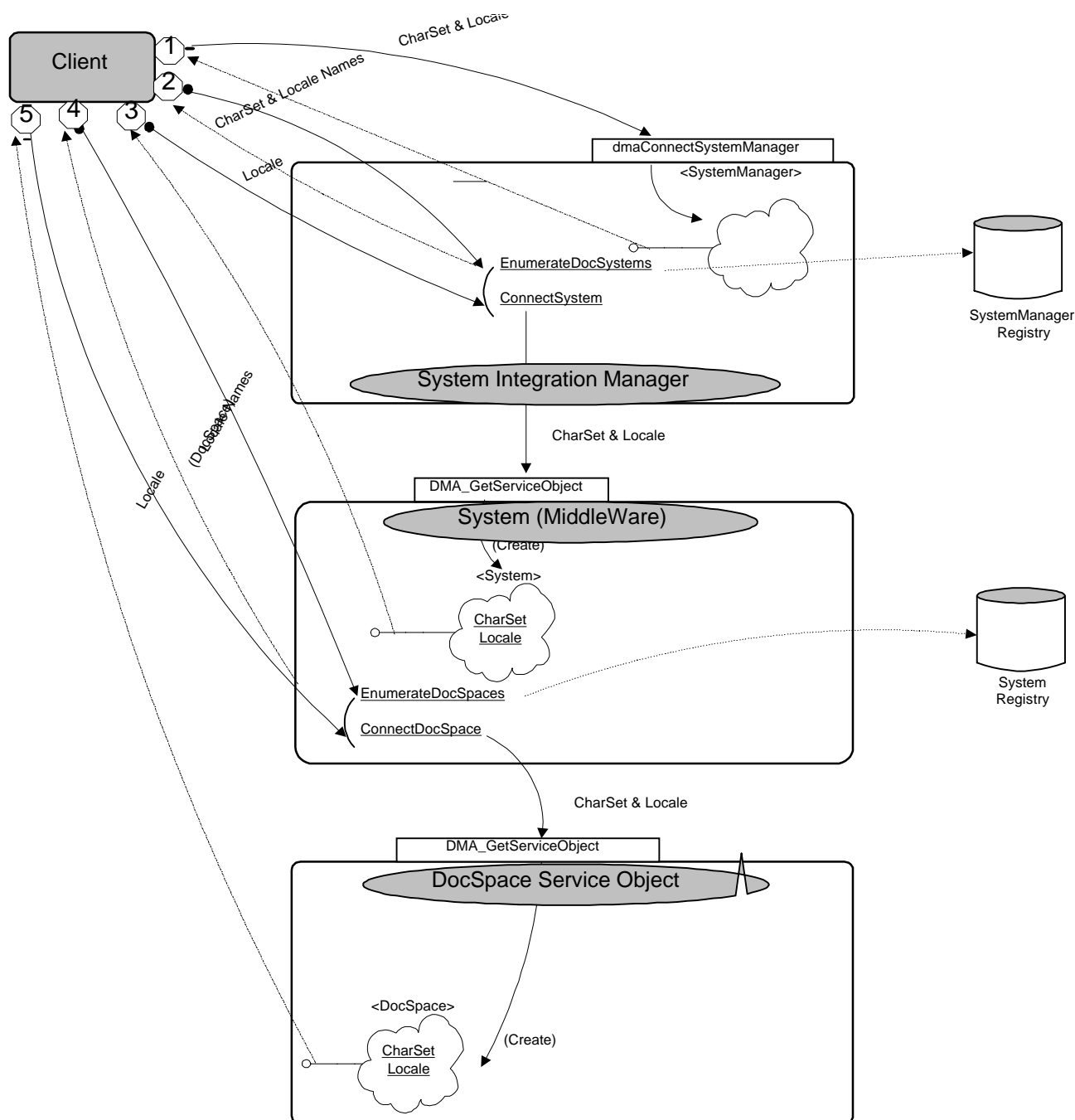


Figure 3-4: DMA Integration Model and Localization Considerations

- 1 The client gets an `IdmaSystemManager` interface pointer by calling a well-known exported `dmaConnectSystemManager` entry point on the System Manager shared library. Via parameters to `dmaConnectSystemManager`,

the client establishes the character set encoding to be used within the context of the System Manager object, and optionally a locale name.

- 2 The client application calls the `IdmaSystemManager::EnumerateSystems()` method to enumerate visible systems. This method returns an enumerator object that supplies unauthenticated System objects.
- 3 Alternatively, the client connects to a system by calling the `IdmaSystemManager::ConnectSystem()` method specifying one of the locale names that is supported by the system. The list of applicable locale names may be established by a prior call to `IdmaSystemManager::EnumerateSystems` and subsequent inspection of the System object's Local Names property.

At this point in time, the system object has been created and its `CharSetEncodingID` and `LocaleName` properties may be interrogated. Their values will correspond to the system's registered character set encoding and the locale name specified to `IdmaSystemManager::ConnectSystem()`.

- 4 The client invokes the `IdmaSystem::EnumerateDocSpaces` method to enumerate the Document Spaces that a DMA system supports. This method returns an enumerator object that supplies unauthenticated DocSpace objects.
- 5 Alternatively, the client invokes the `IdmaSystem::ConnectDocSpace()` method specifying one of the locale names supported by the Document Space. The list of applicable locale names may be established by a prior call to `IdmaSystem::EnumerateDocSpaces` and subsequent inspection of the DocSpace object's Local Names property.

At this point in time, a DocSpace object has been created and its `LocaleName` property may be interrogated. Its value will be the locale name that was specified to the `IdmaSystem::ConnectDocSpace()` method.

Behind the scenes, `IdmaSystem::ConnectDocSpace()` activated the Document Space service object by invoking the `DMA_GetServiceObject()` function, specifying the DMA system's character set encoding and optionally a locale name, which created a DocSpace object and returned the `IdmaDocSpace` COM interface pointer.

3.3.2.5 Authentication

DMA factors authentication into a separate `IdmaAuthentication` interface which may be present on DMA System and DocSpace objects. Typically, user authentication is performed by invoking the `AuthenticateUser` method, which implements an authentication model derived from the ODBC `SQLBrowseConnect` model.

DMA authentication is based upon an iterative stateless exchange of information between the client and System or Document Space, hereafter both are referred to as the *service*. The client supplies authentication parameters to the service via a *DmaString* parameter. If the service requires additional authentication parameters, the service provides the client with a *DmaString* parameter that describes the additional authentication parameters. This exchange typically starts with the client providing the service an empty authentication parameter string and iteratively refining it's authentication parameter string until successfully authenticated or a failure is detected.

DMA currently defines two mechanisms that DMA System and Document Space implementations may use to implement authentication. DMA System and Document Space providers are encouraged to use one or both of these mechanisms instead of inventing equivalent mechanisms. Use of these documented mechanisms will increase interoperability by decreasing the need for out of band agreements between DMA application providers and DMA System and Document Space providers. The authentication mechanism is extensible; DMA System and Document Space providers are free to extend the authentication mechanism as required for their particular need.

3.3.2.5.1 The DMA Authentication Protocol

The DMA Authentication mechanism is best described as a protocol. The client submits a *request message* and the service responds with a *response message*. Each of these messages is encoded in a *DmaString* and syntactically conforms to the following BNF syntax based upon the ODBC SQLBrowseConnect syntax:

```
request-message ::= connection-string
response-message ::= connection-string
connection-string ::= attribute["," connection-string]
attribute ::= [ "*" ] keyword "=" value
keyword ::= dma-keyword | provider-keyword
dma-keyword ::= { "SCHEME" | "UID" | "PWD" | "KRB-VERSION" | "KRB-PRINCIPAL" | "KRB-TICKET" | "KRB-AUTH" } [ ":" localized-identifier ]
provider-keyword ::= identifier [ ":" localized-identifier ]
value ::= "{" value-list "}" | "?" | identifier
value-list ::= identifier [ ":" localized-identifier ][ "," value-list ]
```

identifier and *localized-identifier* consist of one or more characters. *keyword* is to be case insensitive.

An asterisk preceding a keyword indicates that the keyword is optional and need not be present in the next request message.

3.3.2.5.2 DMA Authentication Schemes

The service response to an empty request message indicates which DMA authentication schemes the service supports. DMA 1.0 defines two authentica-

tion schemes and allows providers to extend the set of DMA authentication schemes. The DMA-BASIC scheme provides for a typical user-id and password based authentication. The DMA-KERBEROS scheme provides a mechanism for the client to provide the service with a Kerberos ticket and authenticator.

The following examples illustrate Authenticate method calls by showing input string, output string and return code values.

An initial Authenticate method call is used to determine which authentication schemes are supported:

```
Input string = "" or NULL
Output string = "SCHEME={DMA-BASIC,DMA-KERBEROS,<other>}"
Return Code = DMARC_NEED_MORE_DATA
```

Note: the service responds with one or more schemes. <other> denotes a scheme not standardized by DMA and must not start with the "DMA-" prefix.

3.3.2.5.2.1 DMA-BASIC Authentication

The Client selects the DMA-BASIC scheme:

```
Input string = "SCHEME=DMA-BASIC"
Output string = "SCHEME=DMA-BASIC;UID:UserName=?;PWD:Password=?"
Return Code = DMARC_NEED_MORE_DATA
```

The Client provides user-id and password:

```
Input string = "SCHEME=DMA-BASIC;UID=Joe DMA;PWD=DMA4ME"
Output string = NULL
Return Code = DMARC_SUCCESS
```

The DMARC_SUCCESS return code indicates that the user has been successfully authenticated.

3.3.2.5.2.2 DMA-KERBEROS Authentication

The DMA-KERBEROS scheme supports different versions of Kerberos. A Kerberos scheme authentication message exchange follows:

The client selects the DMA-KERBEROS scheme:

```
Input string = "SCHEME=DMA-KERBEROS"
Output string = "SCHEME=DMA-KERBEROS;KRB-VERSION={4,5,DCE}"
Return Code = DMARC_NEED_MORE_DATA
```

The client selects the DMA-KERBEROS version DCE:

```
Input string = "SCHEME=DMA-KERBEROS;KRB-VERSION=DCE"  
Output string = "SCHEME=DMA-KERBEROS;KRB-VERSION=DCE;KRB-  
PRINCIPAL=DmaSystem@realm;KRB-AUTH=?;KRB-TICKET=?"  
Return Code = DMARC_NEED_MORE_DATA
```

The response supplies the client with a Kerberos principal identifier via the KRB-PRINCIPAL value and requests that the client supply an appropriate Kerberos Authenticator and Ticket via the KRB-AUTH and KRB-TICKET values respectively.

The Client responds with an appropriate authenticator and ticket:

```
Input string = "SCHEME=DMA-KERBEROS;KRB-VERSION=DCE;KRB-  
PRINCIPAL= DmaSystem@realm;KRB-AUTH=<Base-64>;KRB-  
TICKET=<Base-64>"  
Output string = NULL  
Return Code = DMA_SUCCESS
```

where <Base-64> is a Base64 encoding per RFC 1521 of the Kerberos Authenticator and Ticket encrypted values.

3.3.2.5.2.3 Encryption

In DMA 1.0, there is no mechanism to support end-to-end encryption of authentication information using the DMA-BASIC scheme. It is reasonable for a distributed DMA middleware to encrypt the connection strings such that they never appear in the clear on a network. The current model requires that the DMA middleware decrypt the connection string before presenting it to a Document Space Service Object or returning it to a client. As such, the authentication information will at least be transiently stored in the clear by the DMA middleware at the Document Space Service Object and client points of presence.

3.3.2.5.3 Automatic Authentication

DMA 1.0 also provides a mechanism that enables System and DocSpace Service Objects to implement authentication without any caller supplied parameters through the use of the `AutoAuthenticateUser` method. Possible implementations of this method could be predicated upon a client application's environmental state or an earlier successful authentication.

A client application has control over automatic authentication in that it may choose to attempt automatic authentication by invoking the `AutoAuthenticateUser` method or utilize an interactive authentication mechanism by invoking the `AuthenticateUser` method.

3.3.2.1 Identifying and Accessing Persistable Document Space Objects

DMA objects with persistent state have non-null Object Instance ID (OID) property values. DMA OIDs are defined using an Uniform Resource Locator (URL) syntax and are intended to be useable as Internet URLs and can be encapsulated in ODMA Document IDs.

DMA 1.0 OIDs consist of two mandatory GUIDs, one that is a DMA System ID, and another that is a DMA DocSpace ID, along with a document space specific object identifier. If the document space uses GUIDs as its internal object identifier or maintains a GUID association with its internal object identifier, it may expose this Object ID in the DMA OID.

Methods for extracting the DMA System ID, DocSpace ID, and optional Object ID exist on the `IdmaOID` interface which is required to be supported on the System Manager object. These methods can be used along with other DMA methods to instantiate a DMA object given its OID. The following illustrates one possible object instantiation mechanism:

- Get a System Manager object (SysMgr) using `dmaConnectSystemManager`
- Use `SysMgr->QueryInterface(IID_IdmaOID)` to get an `IdmaOID` interface (OID)
- Use `SysMgr->ConnectSystem(Oid->GetSystemId(OID))` to connect to the system (System) that generated the OID. Authenticate yourself if necessary.
- Use `System->ConnectDocSpace(Oid->GetDocSpaceId(OID))` to connect to the document space (DocSpace) that generated the OID. Authenticate yourself if necessary.
- Use `DocSpace->ConnectObject(OID)` to instantiate the DMA object

3.3.2.1.1 DMA OID Syntax Specification

This section defines the syntax of DMA Object Instance IDs (OIDs) using Internet URL conventions and defines how DMA OIDs are to be encapsulated in ODMA Document IDs.

DMA OIDs are considered logically *to* be an Uniform Resource Locator (URL) to a persistent DMA object. A DMA client application can get an in-memory instance of a persistent DMA object when presented with a DMA OID. Implicit in this assumption is that the client application can:

- Can access a DMA Point of Presence (POP) that can connect to the DMA system that generated the OID.

- Can connect to and successfully authenticate itself to the DMA System that generated the OIID
- Can connect to and successfully authenticate itself to the DMA DocSpace that generated the OIID
- Has sufficient authorization to instantiate the DMA object

DMA OIIDs can be easily exchanged by client applications in a manner similar to an HTTP URL. For instance, an application should be able to embed a DMA OIID in an e-mail message or a database. While there has been some discussion that it may be preferable for DMA OIIDs to be presentable in a human readable form, there is no existing requirement for this. Likewise, there is no existing requirement that they be sufficiently short for a human to memorize or manually record, e.g. write it down on paper.

3.3.2.1.2 The DMA URL Scheme

This proposal defines DMA OIIDs as a DMA URL scheme (consistent with RFC 1738) and as ODMA encapsulated DMA URLs. DMA URLs describe the identity of a DMA object and a mechanism exists to locate a DMA object in a hierarchical manner relative to a DMA System and DMA DocSpace. The informal syntax for a DMA URL is:

```
dma://[<dma pop>]/<system id>/<docspace id>/<object id>[:guid=<object guid>]
```

NLS character set issues would have been introduced if the System and DocSpace were encoded as human readable names in the DMA URL. For this reason, the System and DocSpace Dmalds (GUIDs) are utilized instead of the human readable names so as to avoid character set and name stability problems.

Following this line of reasoning, the syntax for the OIID components is:

```
<dma pop> := <host name>
<system id> ::= <Textualized System Dmald>
<docspace id> ::= <Textualized DocSpace Dmald>
<object id> ::= <Printable ASCII per HTTP URL conventions>
<object guid> ::= <Textualized Object Dmald>
```

The <dma pop> (read DMA Point of Presence) is a host name (e.g. library.xyz.com) that can be optionally present in the DMA OIID. Its purpose is to provide a DMA client with a name of an Internet host that most likely can connect to an OIID's DMA System and DMA DocSpace. The host name may include a userid and password per Internet URL conventions. The "file" and proposed "ldap" URL schemes provide precedence for this URL component being optional.

At present, connection to a remote DMA point of presence is not specified using a DMA API or protocol. DMA definition of an application protocol would be required in order to perform this type of connection and as such is outside of the DMA 1.0 scope of effort. DMA 1.0 requires the <dma pop> to not be present in OIIDs generated by DMA DocSpaces and that the <dma pop> field be ignored upon presentation to a DMA 1.0 system.

The <system id> is the DMA GUID used to register the DMA System that generated the OIID. It is considered a hint in that a DMA DocSpace may be accessible from multiple DMA Systems. Any DMA System that supports the DMA DocSpace <docspace id> may suffice for accessing the persistent object referenced by the OIID, however the system <system id> is considered the best choice since it was known to work at some time in the past.

The <object id> is any sequence of data encoded into an allowed subset of ASCII according to the rules governing Internet URLs. The “/”, “,” and “?” characters (and a few others) are reserved for use by the DMA URL scheme and cannot appear in the <object id> (they must be escaped). Beyond the encoding rules, the structure of the <object id> is determined entirely by the underlying Document Space implementation.

If a Document Space maintains a DMA GUID for a particular object, it is encouraged to expose that object GUID in its DMA URL using the optional “;guid=<object guid>” syntax.

3.3.2.1.2.1 DMA URL Examples:

- All URL components specified except object id tag

```
dma://library.xyz.com/01234567-8901-2345-6789-012345678901/  
01234567-8901-2345-6789-012345678901/1000034
```

- <dma pop> omitted per DMA 1.0 conventions

```
dma:/// 01234567-8901-2345-6789-012345678901/  
01234567-8901-2345-6789-012345678901/1000034
```

- An URL with an object guid tag

```
dma:/// 01234567-8901-2345-6789-012345678901/  
01234567-8901-2345-6789-012345678901/  
type=34;guid=01234567-8901-2345-6789-012345678901
```

- An URL with an object guid tag and a trivial <object id>

```
dma:/// 01234567-8901-2345-6789-012345678901/  
01234567-8901-2345-6789-012345678901/  
;guid=01234567-8901-2345-6789-012345678901
```


3.3.2.1.2.2 ODMA Encapsulation

It has been proposed to the ODMA SPT that the vendor name "DMA" be reserved for encapsulation of DMA URLs in ODMA Document IDs. An encapsulated DMA ID conforms to the following syntax:

::ODMA\DMA\<DMA OIID>

This encapsulation enables ODMA Document IDs to be treated by DMA implementations as DMA OIIDs from an input parameter perspective. Any method that takes a pDmaOIID input parameter will handle either the DMA URL form or the ODMA encapsulated DMA URL form. DMA implementations will not generate the ODMA encapsulated DMA URL form of an OIID as a return parameter.

Note: ODMA Document IDs are encoded using the "native character set of the system", e.g. Unicode in Windows 32. DMA OIIDs are always encoded in US-ASCII. As such, character set translation will likely need to occur in an ODMA/DMA implementation above the DMA API.

3.3.2.1.2.3 BNF for DMA URLs

This is a BNF-like description of the DMA OIID Uniform Resource Locator syntax, using the conventions of RFC822, except that "|" is used to designate alternatives, and brackets [] are used around optional or repeated elements. Briefly, literals are quoted with "", optional elements are enclosed in [brackets], and elements may be preceded with <n>* to designate n or more repetitions of the following element; n defaults to 0.

```
dmaurl      = "dma:/" [login] "/" dmasystem "/"
              dmadocspace "/" dmaobjectid [ dmaobjectguid ]
dmasystem   = dmaguid
dmadocspace = dmaguid
dmaobjectid = oidchars
dmaobjectguid = ";guid=" dmaguid
dmaguid     = 1*dmaguidchar
dmaguidchar = hex | "-"
oidchars    = *oidchar
oidchar     = alpha | digit | safe | ":" | "@" | "&" | "="
```

; URL schemeparts for ip based protocols:

```
login       = [ user [ ":" password ] "@" ] hostport
hostport    = host [ ":" port ]
host        = hostname | hostnumber
hostname    = *[ domainlabel "." ] toplabel
domainlabel = alphadigit | alphadigit *[ alphadigit | "-" ] alphadigit
toplabel    = alpha | alpha *[ alphadigit | "-" ] alphadigit
alphadigit  = alpha | digit
```



```

hostname    = digits "." digits "." digits "." digits
port        = digits
user        = *[ uchar | ";" | "?" | "&" | "=" ]
password    = *[ uchar | ";" | "?" | "&" | "=" ]
urlpath     = *xchar

; Miscellaneous definitions
lowalpha    = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" |
             "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" |
             "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" |
             "y" | "z"
hialpha     = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
             "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
             "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

alpha       = lowalpha | hialpha
digit       = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
             "8" | "9"
safe        = "$" | "-" | "_" | "." | "+"
extra       = "!" | "*" | "'" | "(" | ")" | ","
national    = "{" | "}" | "|" | "\" | "^" | "~" | "[" | "]" | "`"
punctuation = "<" | ">" | "#" | "%" | "<>"

reserved    = ";" | "/" | "?" | ":" | "@" | "&" | "="
hex         = digit | "A" | "B" | "C" | "D" | "E" | "F" |
             "a" | "b" | "c" | "d" | "e" | "f"
escape      = "%" hex hex
unreserved  = alpha | digit | safe | extra
uchar       = unreserved | escape
xchar       = unreserved | reserved | escape
digits      = 1*digit

```

3.3.3 DMA Service Objects

DMA System, DocSpace, Text Ordering, OIID Parser and Scope Factory implementations are types of DMA Service Objects. The DMA Service Object model provides a simple operating environment neutral mechanism for registering service providers and a uniform (although platform dependent) mechanism for the System Manager or a service object to obtain instances of other service objects.

DMA is silent about how DMA Service Object implementations are distributed and installed at various points of presence. DMA does specify how a DMA Service Object is registered and publishes its existence at a point of presence and how a DMA Service Object delivers a requested COM interface.

3.3.3.1 Service Object Registration

The DMA System Manager and System objects support the *IdmaServiceRegistry* interface. This interface has methods for registering and publishing service elements. A single program module may provide the implementation for multiple Service Objects each of which are described by a distinct service element.

When registered, each service element is given a distinct *RegistrationID* which can later be used to retrieve, publish, unpublish or remove the service element information. The DMA Service Registry maintains the following information for each service element:

- A *ServiceObjectId* which is a persistent *DmaId* value for the service object. For System service objects this is the System Id, for DocSpace service objects this is the DocSpace ID and for Text Ordering service objects, this is the Collation Sequence ID. For a Scope Factory, it must be the value *dmaServiceObject_ScopeFactory* and for an OIID Parser similarly *dmaServiceObject_OIIDParser*.
- A *ServiceObjectTypeId* which is a *DmaId* indicating which type of service object this service element describes. It may be one of *dmaServiceType_System*, *dmaServiceType_DocSpace*, *dmaServiceType_TextOrdering*, *dmaServiceType_ScopeFactory* or *dmaServiceType_OIIDParser*.
- A *Profile* string which will be supplied to the service object implementation during activation and may be used by the service object implementation to locate additional configuration information.
- A *ModuleTypeId* which is a *DmaId* indicating what type of Module implements this service object and, by implication, the means by which an instance of the object can be obtained. DMA 1.0 defines a module type for Windows 32 DLLs. Other module types for Unix shared libraries could easily be added.

- A *ModuleLocation* which is a string describing the location of the Module implementing the service object. The form of this string is dependent upon the *ModuleTypeID*. For the Win32 DLL module type, it will be the pathname of a DLL containing the service object implementation.
- A *CharSetEncodingId* which describes which character set encoding this service object implementation supports.

Registering a service element makes that service element registry information available at the point of presence where the service element was registered (the point of service). Publishing the service element makes the service element registry information available at all points of presence which can reach the point of service.

3.3.3.2 Service Object Activation

Activation of a DMA service object requires that the service object's module be loaded and that a function in that module be invoked, returning the desired object. These steps will vary according to the type of module in which the service object is implemented. DMA System Manager and System objects must provide the implementation of this activation logic.

For the Win32 DLL module type, the DLL referenced by the *ModuleLocation* parameter will export the *DMA_GetServiceObject* function as described below. It is likely that on other platforms a similar mechanism (e.g. using shared libraries) will be defined and that these will employ an entry point with the same general signature. Other module types and activation mechanisms may also be defined, for Win32 as well as for other platforms.

The general signature for the *DMA_GetServiceObject* function is as follows:

```
DmaRC DMA_GetServiceObject (
    pDmaVoid pvEnvInfo,
    pDmaOpaqueString pProfile,
    Dmapv pIMalloc,
    Dmapv punkParent,
    DmaInteger32 lCharSetEncodingID,
    pDmaString pLocaleName,
    pDmaId pServiceObjectTypeID,
    pDmaId pServiceObjectID,
    DMA_REFIID riid,
    Dmappv ppIServiceObject );
```

This function is invoked using *Profile*, *CharSetEncodingID*, *ServiceObjectTypeID*, and *ServiceObjectID* parameter values extracted from a Service Registry. The values for the other parameters are generally derived from parameters specified on the *dmaConnectSystemManager* function and the *IdmaSystemManager::ConnectSystem* and *IdmaSystem::ConnectDocSpace* methods as appropriate.

A successful DMA_GetServiceObject call returns the COM interface specified by the *riid* parameter on a service object instance.

3.3.3.3 The System Object View: Integrating Systems for Access

3.3.3.3.1 Installing a System Object at the point of access

The installation of a DMA System service object is performed by a DMA application program operating with a connection to a DMA System Manager. The installation application registers and publishes the System service object using the DMA System Manager's IdmaServiceRegistry interface, after which time, clients of that DMA System Manager may connect to the newly installed System using the IdmaSystemManager::ConnectSystem or IdmaSystemManager::EnumerateSystems methods.

3.3.3.3.2 Typical Connection Entry into the System Service Object Implementation Program

System Service Object implementations are required to honor DMA_GetServiceObject requests for the IdmaSystem interface. The IdmaSystemManager::ConnectSystem method simply activates the System service object and calls its DMA_GetServiceObject function passing downward its *riid* input parameter.

3.3.3.4 The DocSpace Object View: Integrating DocSpace Objects into Systems

3.3.3.4.1 Installing a DocSpace Service Object at a Point of Service

The installation of a DMA DocSpace service object is performed by a DMA application program operating with a connection to a DMA System. The installation application registers and publishes the DocSpace service object using the DMA System's IdmaServiceRegistry interface, after which time, clients of that DMA System may connect to the newly installed DocSpace using the IdmaSystem::ConnectDocSpace or IdmaSystem::EnumerateDocSpaces methods.

3.3.3.4.2 Typical Connection Entry into the DocSpace Service Object Implementation Program

DocSpace Service Object implementations are required to honor DMA_GetServiceObject requests for the IdmaDocSpace interface. The IdmaSystem::ConnectDocSpace method simply activates the DocSpace service object and calls its DMA_GetServiceObject function passing downward its *riid* input parameter.

3.3.3.5 The Text Ordering Object View: Integrating Text Ordering Objects for Access

3.3.3.5.1 Installing a Text Ordering Service Object at a Point of Service

The installation of a DMA Text Ordering service object is performed by a DMA application program operating with a connection to a DMA System Manager. The installation application registers the Text Ordering service object using the DMA System's IdmaServiceRegistry interface. Text Ordering Service Objects are not directly accessible by client applications. They are typically utilized by System Service Objects which utilize their parent object's (System Manager) registry to locate a text ordering service element offering a needed text ordering.

3.3.3.5.2 Typical Connection Entry into the Text Ordering Service Object Implementation Program

Text Ordering Service Object implementations are required to honor DMA_GetServiceObject requests for the IdmaTextOrdering interface.

3.3.3.6 The Scope Factory Object View: Integrating a Scope Factory Object for Access

3.3.3.6.1 Installing a Scope Factory Service Object at a Point of Service

The installation of a DMA Scope Factory service object is performed by a DMA application program operating with a connection to a DMA System. The installation application registers the Scope Factory service object using the DMA System's IdmaServiceRegistry interface. Scope Factory Service Objects are not directly accessible by client applications. They are accessed indirectly through the IdmaScopeFactory interface of the System object through which they are registered.

3.3.3.6.2 Typical Connection Entry into the Scope Factory Service Object Implementation Program

Scope Factory Service Object implementations are required to honor DMA_GetServiceObject requests for the IdmaScopeFactory interface.

3.3.3.7 The OIID Parser Object View: Integrating OIID Parser Objects for Access

3.3.3.7.1 Installing a OIID Parser Service Object at a Point of Service

The installation of a DMA OIID Parser service object is performed by a DMA application program operating with a connection to a DMA System Manager. The installation application registers the OIID Parser service object using the DMA System's IdmaServiceRegistry interface. OIID Parser Service Objects are not directly accessible by client applications. They are accessed indirectly through the IdmaOIID interface of the System Manager object with which they are registered.

3.3.3.7.2 Typical Connection Entry into the OIID Parser Service Object Implementation Program

OIID Parser Service Object implementations are required to honor DMA_Get-ServiceObject requests for the IdmaOIID interface.

3.4 DMA Distribution Model

3.4.1 Client-Server Operation

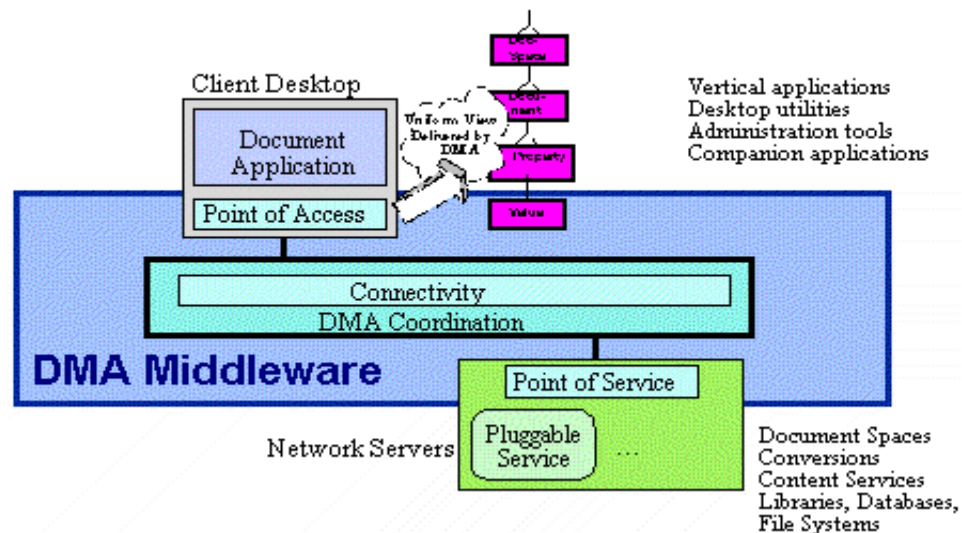


Figure 3-5: Basic Client-Server Distribution Model

Access to a DMA system is provided through symmetrical points of access (where clients interact with the DMA system) and points of service (where service elements are operated under the DMA system). DMA-compliant middleware manages the distribution of access, so that clients and servers can be located on the same platform, on different systems on a network, on different networks, or at remote points of an enterprise or inter-enterprise wide-area network. From the client point of view, DMA provides a uniform view of all types of documents, regardless of location, creation method, etc.

Although we tend to think of clients and services as residing on different platforms over a network for DMA it is valuable to think of both client applications and service elements as all residing at a DMA point of presence.

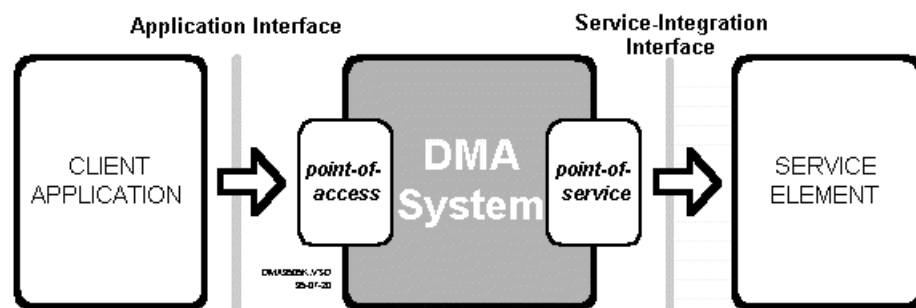


Figure 3-6: Basic Point-of-Presence Model

In the point-of-presence model, three elements are potentially resident on the same platform:

- Client application software making access to documents and collections of documents via the DMA application interface to a DMA system
- Service-element software that offers access to documents and collections by operating under the service-integration interface of a DMA system
- DMA system software that provides for integration of the local platform's DMA-compliant applications and service elements into distributed operation.

3.4.2 Basic DMA Distribution

DMA manages distributed services in several ways, as shown in the diagram below. In the simplest case of a local Windows-based service element, for example, local coordination is used. Interoperability with other platforms can be handled in this way, through a Windows front end.

For services that are distributed across a Local Area Network (LAN), DMA Systems provides distributed coordination to transparently manage the network connectivity that is required.

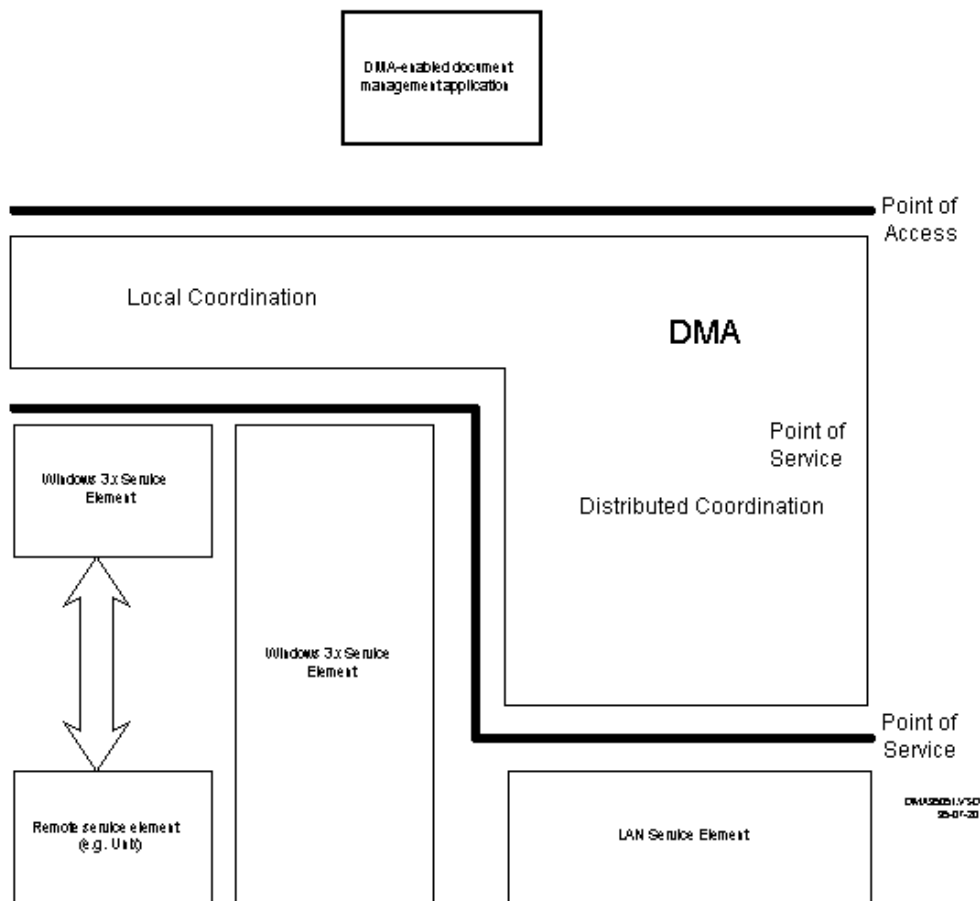


Figure 3-7: Varieties of Distribution at a Point-of-Access

3.5 DMA Query Model

3.5.1 Introduction

Query (i.e., associative retrieval), especially across multiple document spaces, is the central mechanism of DMA to retrieve the properties of independently persistent objects.

Document spaces support collections of documents. These collections have some organizing principles that are appropriate and valuable in support of the intended use of the documents. In DMA, the organization of the documents in a document space (i.e., the metadata) is available for examination. Every DMA document space "publishes" its metadata and the types of query operations it supports. Not every document space need support the same search technology or organization of objects. However, whatever technology and organization that is used is expressed through DMA in a uniform way. That is, just as different document spaces can organize their objects in various ways, how those objects are explored and distinguished by searching may also vary. Nevertheless, applications have a uniform way to determine the search capabilities of any document space.

A Document Space Scope object supplies information about searchable classes and query capabilities. Given a scope object, the following questions can be answered:

- What are the searchable classes of persistent objects in the document space?
- What are the properties of the searchable classes that are available for query formulation?
- What are the query operators that are supported?
- What properties can be selected for presentation in a Query Result Row?
- What options are available for the ordering the Query Result Rows produced by a query?

The following discusses how scopes are used to answer these questions.

3.5.2 Scope Object

All search operations are constructed and initiated using a scope object. Each document space can supply a scope object. Obtaining the desired scope object is the first step in performing a query.

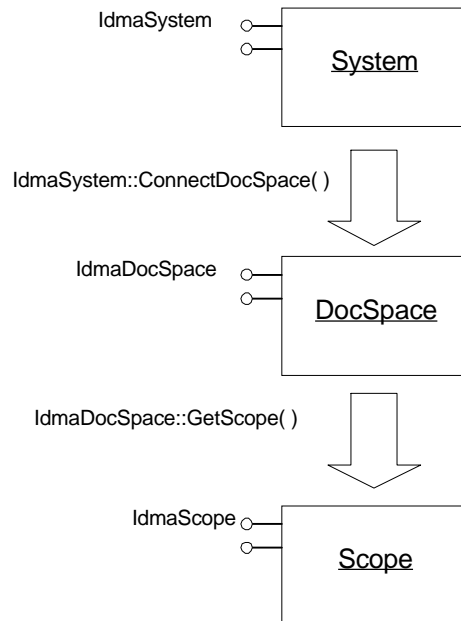


Figure 3-8: Creation of a Document Space Scope object

One or more Scope objects can be combined into a new merged Scope object by calling the `IdmaScopeFactory::CreateScope` method on the DMA System object. That is the scope merging mechanism. A merged Scope object has the same interface as an individual document space Scope object, but it provides a combined view of the metadata of its component scopes. A merged Scope object transparently distributes (possibly massaged) query requests to each of its component Scope objects, and the Query Result Set object it returns delivers a merged set of Query Result Rows from the Query Result Set objects of the component Scope objects.

An application can always use a document space Scope object directly to perform queries. There is no requirement that a merged Scope object be used to perform query operations.

3.5.3 Basic Search Formulation Process

A typical scenario for formulation of a query is the following:

- Create a target Scope object: Either (1) Create an individual document space Scope object by calling `IdmaObjectFactory::CreateObject` on the document space object. Or (2) create a set of individual document space Scope objects as in (1), and then create a merged Scope object designat-

ing them as its component Scope objects by calling `IdmaScopeFactory::CreateScope` on the System object.

- Use the target Scope object's properties to find the classes of objects that can be searched, the properties of the searchable classes that can participate in a query and that can be delivered as results of a query, the kinds of query operations that can be performed, and how properties can be used in ordering the Query Result Rows of the query.
- Use this information to construct a Query object and its dependent Query Node objects based on user input.
- Call the Scope object's `IdmaScope::ExecuteSearch` method, passing it the Query object in order to set up the query. A Query Result Set object is returned by `ExcuteSearch`.
- Repeatedly call the `IdmaResultSet::GetNextResultRow` method on the Query Result Set object to get each Query Result Row object.

A variation is to save (for example, in a file) the data in the Query object and all its dependent objects by serializing all their properties, and later reuse the saved query data to recreate the Query object and all of its dependent objects.

3.5.4 The ExecuteSearch Method

A query is initiated by calling the `ExecuteSearch` method in the `IdmaScope` interface on the Scope object. This method takes, as input, a description of the query to perform and produces a Query Result Set object as output. The Query Result Set object can be used to generate the collection of Query Result Row objects that conform to the constraints of the query. The `ExecuteSearch` method does not keep a pointer to the Query object or any of its dependent subobjects. Thus, changes to the Query object or any of its subobjects after the `ExecuteSearch` method has returned will have no effect on the execution of the query. It is an error to change the Query object or any of its dependent subobjects during the execution of the `ExecuteSearch` method, and the results of doing so are unpredictable.

The `ExecuteSearch` method has the following signature:

```
DmaRc IdmaScope:: ExecuteSearch(  
  
    Dmapv          pIQueryObject,  
    Dmapv          pICallback,  
    DmaBoolean     bRequestElimination,  
    DMA_REFIID     riidResultSet,  
    pDmapv         ppIResultSet);
```

The parameters of the `ExecuteSearch` method are the following:

- **plQueryObject**
This is a pointer to the query object. The query object is the root of a tree of objects that specify the details of the query.
- **plCallback**
This is a pointer to a callback interface or NULL. The ReportProgress method of the callback interface might (at the option of the implementation) be called at intervals during the execution of the ExecuteSearch method.
- **bRequestElimination**
This is a Boolean parameter that, if TRUE, requests the scope to use three valued elimination to deal with classes, properties, and operators it doesn't fully understand. (Three valued elimination is a generalization of ANSI standard SQL three valued logic, and is explained below.) The client should always pass FALSE for this parameter. Then ExecuteSearch will return errors for undefined properties and classes in the Query object, since these are query construction errors committed by the client.
- **riidResultSet**
ExecuteSearch returns a Query Result Set object in *ppIResultSet. This parameter specifies the desired interface on the Query Result Set object.
- **ppIResultSet**
ExecuteSearch returns a Query Result Set object in *ppIResultSet. The Query Result Rows are enumerated by calling the GetNextResultRow method of the IdmaResultSet interface of this object.

The ExecuteSearch method on an individual document space Scope object initializes the query, but may or may not synchronously retrieve or initiate asynchronous retrieval of any Query Result Rows. Retrieval of Query Result Rows may optionally be postponed until the GetNextResultRow method is called on the Query Result Set object. Query Result Rows may all be retrieved at once and buffered internally, or may be retrieved incrementally either singly or in batches as GetNextResultRow is called. The choice of retrieval policy is left to the implementation of ExecuteSearch.

The ExecuteSearch method on a merged Scope object calls the ExecuteSearch method of all of its component scopes before returning.

More detail for ExecuteSearch can be found in the interfaces reference section.

3.5.5 Query Result Sets

The Query Result Set object is the primary output parameter of IdmaScope::ExecuteSearch.

The objects produced by a Query Result Set object are called Query Result Row objects. The Query Result Set object generates Query Result Row ob-

jects that satisfy the search request. Each Query Result Row object has the properties that correspond to each property selected in the Selections list property of the main Query Root object.

Sequencing through the Query Result Row objects is performed by calling the `IdmaResultSet::GetNextResultRow` method on the Query Result Set object.

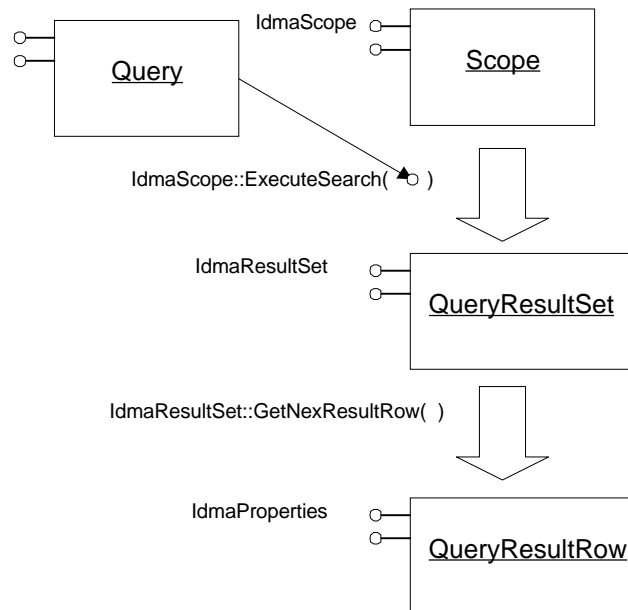


Figure 3-9: Execution of a Query

3.5.6 Scopes and Their Creation

A Document Space Scope object provides the interface to the query engine of a single document space. A merged Scope object provides a unified interface to one or more document space query engines.

Any Scope object provides the metadata required to construct a query against itself. It also provides the `IdmaScope` interface, which includes the `ExecuteSearch` method. The `ExecuteSearch` method returns a Query Result Set object, which can be used to return the Query Result Row objects satisfying the query.

3.5.6.1 Properties of Scopes

A Scope object has a `Searchable Class Descriptions` property that is a list of Class Description objects for each class of objects that can be searched in a query operation using the Scope object. In general metadata spaces must preserve the partial inheritance order of all of the DMA-defined classes it contains. However, the partial ordering of the classes can not always be pre-

served in a merged scope under the union option (see Merging Metadata below). Therefore, a merged scope is permitted (but not required) to have the searchable classes appear as immediate subclasses of class DMA.

The Scope also has an Operators property that is a list of Operator Description objects. Each element of this list is an object that describes a query operator supported by the Scope object. A Query Operator Description object has properties that define which operator is being described, the data type of the value produced by the operator, a description of the operands required by the operator, etc.

The Class Descriptions in the Scope's Searchable Class Descriptions list provide search information. In particular, the Property Description objects in the Properties list of these Class Description objects provide information that includes the following:

- Is Searchable: Indicating whether a property can be used in the Query Expression, and thus can be the basis of a search.
- Is Selectable: Indicating whether the property can be selected as a property of a returned Query Result Row object.
- Query Operator Description list: Containing the Operator Description objects of query operators of which this property can be an operand.
- Is Orderable: Specifying whether this property can be used to help control the ordering of the Query Result Rows of a query.

Class Description objects have a property, Has Include Subclasses, that specifies whether subclass searches are supported by the current class. If subclass searches are supported, then a query can be constructed that specifies that rows from the class and all of its subclasses are to be returned in the Query Result Rows. For example, if the class is Legal Documents with subclasses Depositions and Contracts, the query would return Query Result Rows from all three of these classes, as long as they satisfied the search condition. If the class is DocVersion, then the search spans all document classes.

Class Description objects have another property, Has Proper Subclass Properties, that specifies whether properties from the subclasses can be mentioned in the search condition. For example, continuing the Legal Documents example with subclasses Depositions and Contracts, if Has Include Subclasses were DMA_TRUE, then the query condition could mention properties from the classes Depositions and Contracts, even though the only class mentioned in the From Expression of the query contained was Legal Documents. The Proper Subclass Property Descriptions list property of the Class Description object is a list of the property descriptions for all the subclasses of the current class. This property is important: A property could be included in multiple subclasses, and have slightly different characteristics in each one. The Prop-

erty Descriptions in the Proper Subclass property Descriptions list resolves how such properties should be treated in “include subclasses” queries.

Thus, if Has Include Subclasses and Has Proper Subclass Properties were both DMA_TRUE for the DocVersion class, searches could be made across all searchable document classes using any of the properties of any document class in the search condition.

The Query Construction Class Descriptions list property has Class Descriptions for all classes needed to construct a Query object and any of its subobjects. This includes Query, Query Node and all its subclasses, List Of Object, List Of Binary, List Of Boolean, etc.

The Scope also has a Collation Sequence Ids property, which is a list of the Ids of the collation sequences supported by the scope.

A document space Scope object uses the logical connection of the Document Space object from which it was generated to access the document space query engine. However, the Scope object itself does not supply the IdmaConnection interface.

If for some reason the connection to the document space is permanently lost (i.e., due to an unavailable host), then the Scope object cannot be used for issuing ExecuteSearch. In this case, a new Document Space object and Scope object must be constructed.

All metadata spaces are stable, including the metadata space of a Scope object. Thus, the Scope's metadata cannot be “refreshed”. Instead, a new Scope must be constructed. Client applications can rely on the stability of metadata.

3.5.7 Query Objects

This section describes the properties of the Query object and its subtree of dependent objects. The Query object has properties for the search condition, plus properties that affect the execution of the query, but that have nothing to do with the search condition. More detail on the Query object and its dependent objects can be found in the reference section on objects and properties.

The properties introduced by the Query object include the following:

- Query Root object
- Batch Size Hint
- Maximum Result Items
- Time Limit
- Collation Sequence Id

The Query Root object is the search condition, and is represented by a sub-tree of dependent objects.

The Batch Size Hint property is for Scopes that perform incremental generation of Query Result Rows as opposed to generating all the Query Result Rows before returning any. It is the number of Query Result Rows to get as a unit and save in buffers. The hint may be ignored by the implementation.

The Maximum Result Items property give a hint as to the maximum number of Query Result Rows each document space scope should generate. The hint may be ignored by implementations. A document space may or may not have a default value for this property.

The Time Limit property gives a hint as to the maximum number of elapsed seconds to allow for execution of the query. This the time between the call on IdmaScope:: ExecuteSearch, and the last call on IdmaResultSet:: GetNextResultRow that may return a Query Result Row object. Implementations may choose to ignore this hint. A document space may or may not have a default value for this property.

The Collation Sequence Id specifies which collation sequence all Scope objects involved in the query will use.

The Query Root object is now described in detail.

3.5.7.1 Query Root Object

The Query Root object is the root of a tree of dependent objects that specify the main search condition. If the user specified the query using a query language (as opposed to user interface gestures, e.g. pulldown boxes, etc.), then the Query Root object tree may be thought of as a parse tree plus some other properties. If there is a subquery, then one of the dependent objects is another Query Root object which itself is the root of another dependent tree of objects, etc.

The parse tree paradigm has numerous benefits. It decouples the client software from the query engine software in the document spaces. It makes the DMA API independent of the user interface. It makes DMA independent of the query language used by the client, or even if a query language is used at all as opposed to some other approach, e.g. form based UI's. (It should be noted that due to SQL's importance, the parse tree must fit very well with SQL. The parse tree must also fit well with various different content based retrieval approaches, and query languages other than SQL.) The parse tree paradigm also aids software upgrades to the computers on the network. (Older software can deal more gracefully with unknown or extended parse tree objects from future software releases, than older parsers can deal with future query language syntax extensions.) The parse tree paradigm is extremely extensible, which is critical. Finally, the parse tree paradigm has run time efficiency advantages.

The Query Root object includes the following native properties:

- From Expression
- Selections
- Query Expression
- Orderings
- Distinct Rows Requested

The Query Root object inherits the Operands property from the Query Node class.

The properties of the Query Root object will now be described in the above order.

3.5.7.1.1 Subqueries

A Query Root object (and descendents) may appear as an operand of an operator in the Query Expression subtree of a superior Query Root, forming a subquery. The subquery Query Root may have zero, one or more elements in its Selections list, depending on the operator for which the subquery is an operand.

In considering the Operand Description which admits a subquery the following rules apply:

- A subquery is considered to be an expression. Thus the Allows Expression property of the Operand Description must be TRUE.
- If the Selections list of the subquery is empty, the subquery is considered to generate a singleton result of data type Class. Thus the Allows Singleton property of the Operand Description must be TRUE and the Operand Data Type Property must specify DMA_DATATYPE_CLASS (or DMA_DATATYPE_ANY). The Exists operator is the only operator defined in DMA 1.0 which accepts such an operand.
- If the Selections list of the subquery contains exactly one element, the subquery is considered to generate a list result of data type corresponding to the data type of the select property. Thus the Allows List property of the Operand Description must be TRUE and the Operand Data Type property value must match with the data type of the selection property (or must be DMA_DATATYPE_ANY). The various "In" operators specified in DMA 1.0 accept such an operand.
- If the Selections list of the subquery contains more than one element, the subquery is considered to generate a list result of data type Object. Thus the Allows List property of the Operand Description must be TRUE and the

Operand Data Type property must specify DMA_DATATYPE_OBJECT (or DMA_DATATYPE_ANY). There are no well-known operators specified in DMA 1.0 that accept such an operand.

3.5.7.1.2 From Expression

The value of the From Expression property is the root of a tree of dependent objects that includes an object for each searchable class involved in the query, and the conditions which relates those searchable classes. (In SQL, these conditions are called "join" conditions.)

Since, as in SQL, a searchable class may occur more than once in a From Expression object tree, the client must assign a unique non negative integer to each searchable class object in the From Expression object tree. The properties in the Selections, Query Expression, and Orderings use the value of this number to refer back to the particular instance of the searchable class in the From Expression object tree to which they belong. This integer valued property of Query Searchable Class objects is called Searchable Class Occurrence.

The From Expression object maps well to the SQL 92 "from" clause. The From Expression corresponds to a basic subset of what the SQL 92 standard allows. A simple linear list of searchable classes as in the SQL 89 standard does not give enough control to the query composer as to how classes are joined. Plus, SQL 89 leaves it to proprietary extensions to specify the type of join desired (e.g., inner, left outer, etc.). The From Expression eliminates these problems, is in line with industry standard practice, and makes it easier for non-SQL based query engines to get a handle on what classes are being joined and in what order.

In order to understand how searchable classes are related in the From Expression, one must understand something about the various types of joins in SQL. There are at least the following types of joins in SQL:

- INNER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN
- CROSS JOIN

Consider equi-joins, i.e., joins in which a property in the first searchable class has a value equal to a property in the second searchable class. The inner equi-join is often what is wanted.

However, what happens when one of the properties is null, say, the property for the rightmost searchable class? The answer for inner joins is that no row is returned in the answer set. This is sometimes undesirable:

Suppose the first searchable class is "Specs", a subclass of DocVersion, with properties OIID and SpecNumber. Suppose the other searchable class is "Authors", with properties SpecOIID, Name, Address, and PhoneNum. Suppose the query was "get the spec whose SpecNumber is 1234 and also get the information about all its authors". Then, you want to join the row in Specs to the rows in Authors on the condition "Specs.OIID = Authors.SpecOIID".

Suppose that you enter information about a particular spec. with spec number 1234 into the document database, but at the time that you enter it, you don't know anything about the authors. Then using an inner join in your query would give you no Query Result Row. Most query authors wouldn't like that. They would want to get the row from Specs even though it doesn't join to anything in Authors. This is why the left outer join was invented. It does exactly that.

The right outer join does what the left outer join does, but from the point of view of the second searchable class.

The full outer join is a combination of a left outer join and a right outer join.

Finally, the cross join is simply the Cartesian product. Generally, a full blown Cartesian product is seldom, if ever, what is desired, but that's what SQL gives you if you mention more than one table in the "from" clause, and don't do a join. For Cross Joins, the last operand of the Query Join Op object is null.

In DMA, joins are performed only within document spaces, never across document spaces.

It is possible for the From Expression object to consist of a single searchable class operand. In that case the searchable class is not joined with any other.

If more than one searchable class is involved, then the searchable classes will be joined together, and the value of the From Expression property is a Query Join Op object that is the root of a tree of dependent objects. For example, suppose searchable class S1 is to be joined to searchable class S2, and the properties involved are S1.P1, and S2.P2. Suppose the P1 and P2 properties are integers. Further suppose that the type of join is "inner join". Using SQL 92 notation, this could be written as "S1 INNER JOIN S2 ON S1.P1 = S2.P2". That would be represented by the following From Expression object tree:

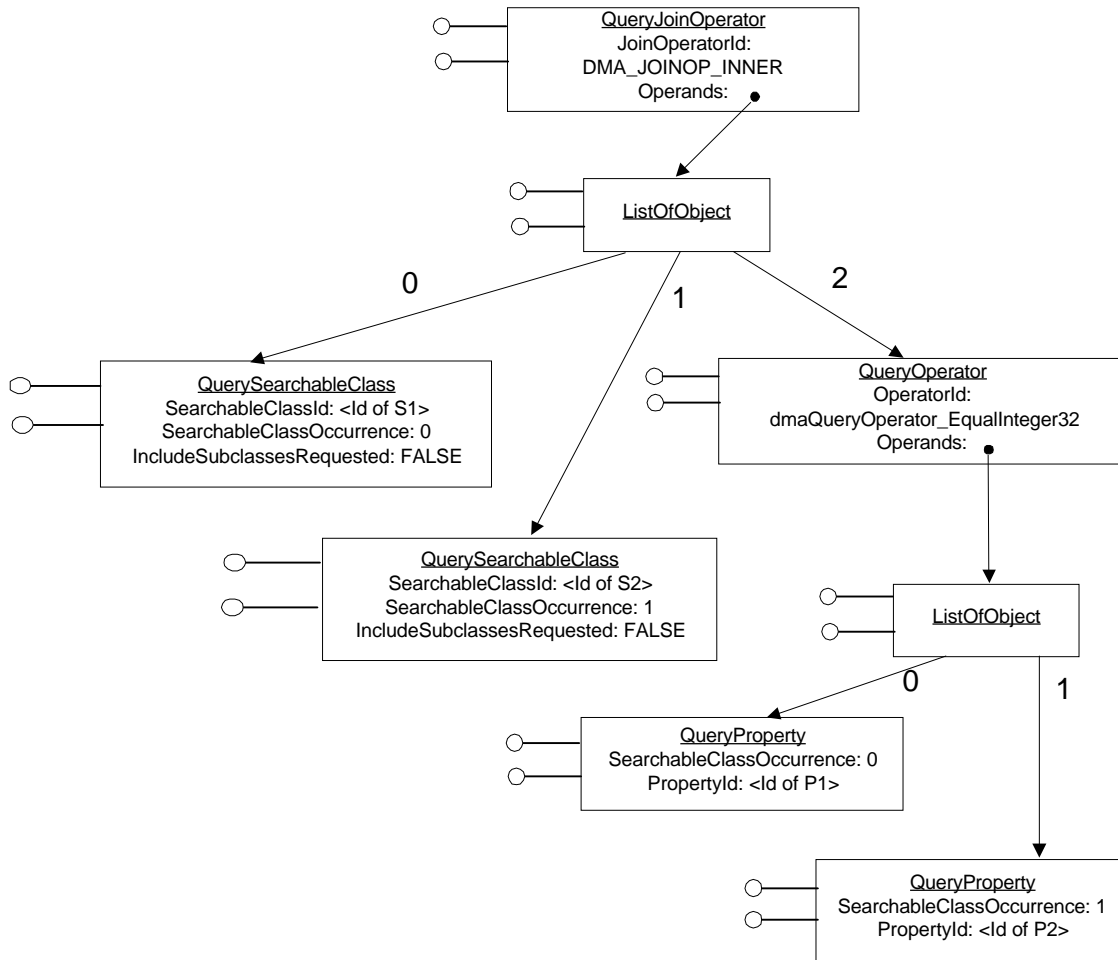


Figure 3-10: Example "From Expression"

The class of the From Expression object must be either Query Searchable Class or Query Join Operator. In the first case, no join is involved, and there is only one node in the From Expression object tree. In the second case, at least one join is involved. Only the first operand of the Query Join Operator object may be another join Query Join Operator object. The second operand must be a Query Searchable Class object. This means, that the From Expression tree can only extend down its leftmost branch. Thus, the joins are performed in a simple linear sequence.

Sometimes, e.g., in the case of the EXISTS operator, it is necessary to specify additional joins in the Query Expression. These joins specify conditions of the form "S1.P1 = S2.P2", just as for "ON" conditions in the From Expression. These joins are inner joins.

3.5.7.1.3 Selections

The purpose of the Selections property is to designate the Property values to be included in each Query Result Row.

The Selections property is object valued. It is a list of objects of class Query Node. All the list elements must be of class Query Property. Each property in the list must be a property of a searchable class in the Scope object's Searchable Class Descriptions list. Furthermore, each property in the list must have the value DMA_TRUE for the Is Selectable property of its Property Description object.

Note that, as previously explained, the value of the property Searchable Class Occurrence of Query Property objects refers back to a unique occurrence of a Query Searchable Class object in the From Expression of the Query Root object.

The Selections list must not be null for the main query. It must be null for subqueries that are operands of the EXISTS operator. The Selections list must be non-null for subqueries under the IN operators, and, it must contain exactly one element of class Query Property, which is a base datatype property of a persistent object.

3.5.7.1.4 Query Expression

The Query Expression property is the root of a tree of dependent objects that specify a Boolean search condition. If the Query Expression property is NULL, then no constraint is put on the persistent objects to be selected. If the Query Expression property is not NULL, then it is the root of an operator/operand subtree of Query Node objects that specifies a Boolean condition. If the value of the Boolean condition is DMA_TRUE, the current objects under scan are returned in the Query Result Set. Otherwise, they are not.

The issue of partially defined expressions can arise when queries are done with merged Scope objects, and when some properties have no value (i.e., are NULL).

3.5.7.1.4.1 Query Execution With Merged Scopes

Consider the case that the Scope object involved in the query is a merged Scope object, as opposed to a Scope object for an individual document space. The metadata of the component Scope objects is merged together to form the metadata for the merged scope object. There are two merge options (which will be discussed in detail below) - union and intersection. The intersection option retains only metadata common to all the component scopes. The union option takes the set union of the metadata of the component scopes. Thus, under the union option, the metadata of the merged scope may have searchable classes, properties, and query operators that are not defined in some of the component scopes.

The `IdmaScope::ExecuteSearch` method of the merged scope object passes the query on to each of the component Scope objects by calling their `IdmaScope::ExecuteSearch` methods. It saves the Query Result Set objects returned by the component scopes and enumerates all the Query Result Rows. It merges the Query Result Row objects and returns them through the Query Result Set object returned from the original `ExecuteSearch` call on the merged Scope.

Thus, it can be seen that in the case of the union option for merging metadata, if the query passed on to the component scopes is not modified, the `ExecuteSearch` method of a component scope can be passed a Query object containing classes and properties that are not defined in that component scope. This gives rise to the issue of partially defined expressions.

3.5.7.1.4.2 Partially Defined Query Expressions

It is important to note that partially defined expressions can arise in Query Expressions even if all classes, properties, and operators are completely defined. For example, consider the Query Expression " $X / Y > Z$ ". Suppose the value of Z is zero in the current row under scan by the query engine of a document space. What happens, you might ask? As another example, consider the same query expression, only this time, in the current row under scan, Z is non zero, but X has no value (alternatively, one could say that X is NULL). What happens, you might ask?

The answers to these questions in ANSI standard SQL are as follows.

If any part of an arithmetic expression is undefined, the whole arithmetic expression is considered undefined. This includes division by zero, NULL properties, etc.

If any part of a string expression is undefined, the whole string expression is considered undefined. This includes NULL properties, etc.

For operators that can produce a Boolean value, a third truth value is added called UNKNOWN. In SQL, there are manifest constants that can be used in queries for all three truth values: TRUE, FALSE, and UNKNOWN. This is referred to as "three valued logic". If part of the expression is not defined, but it doesn't matter, then TRUE or FALSE is returned. If the undefined part of the Boolean expression can affect the truth value of the expression, then the expression evaluates to UNKNOWN.

A relational operator (e.g., " $>$ ", " $>=$ ", " $<$ ", " $<=$ ", " $=$ ", " $!=$ ", etc.) returns UNKNOWN if either of its operands is undefined. (Note that this implies that " $x = x$ " evaluates to UNKNOWN if x is NULL.)

The logical operators AND, OR, and NOT can return TRUE, FALSE, or UNKNOWN. The logical operator evaluates to TRUE or FALSE if the partially de-

defined expression matters to the value of the expression. Otherwise, it evaluates to UNKNOWN.

For "A AND B", if A or B or both is FALSE, the value of the expression is FALSE. If A is TRUE, the value of the expression is B. If B is TRUE, the value of the expression is A. Thus, for example, if A is TRUE, and B is UNKNOWN, then the value of "A AND B" is UNKNOWN.

For "X OR Y", if X or Y or both is TRUE, the value of the expression is TRUE. If X is FALSE, the value of the expression is Y. If Y is FALSE, the value of the expression is X. Thus, for example, if X is FALSE, and B is UNKNOWN, the value of "X OR Y" is UNKNOWN.

For "NOT Z", if Z is TRUE, the value is FALSE, and if Z is FALSE, the value is TRUE. If Z is UNKNOWN, the value of "NOT Z" is UNKNOWN.

Query Expressions are Boolean expressions. (The terminology "Boolean expression" is used by SQL, even though there are three truth values, not just the traditional two.)

The rule in ANSI standard SQL that determines whether the current row under scan is included in the result set or not is this: The current row (or set of joined rows) under scan is returned as one of the Query Result Rows if the truth value of the Query Expression is TRUE. Otherwise, the row is not returned. In other words, if the Query Expression evaluates to UNKNOWN or FALSE, the row (or set of joined rows) is omitted from the Query Result Rows.

In order to deal with performing queries on merged scopes formed under union rules, DMA query adopts the standard three valued logic of ANSI SQL, and extends it in the obvious way to undefined classes and properties. Thus, for example, in a Query Expression, if a property is undefined, it is usually treated as if it were defined but NULL when a copy of the Query object is massaged. This massaging of Query objects to eliminate undefined classes and properties is referred to as "three valued elimination".

The rules of three valued elimination are developed by making the obvious and straightforward extension of the philosophy of three valued logic. In three valued elimination, for arithmetic operators and other operators in the Query Expression whose return value is of one of the DMA base datatypes other than Boolean, an undefined property anywhere in the subexpression below the operator results in the closest ancestor relational operator node above the operator being replaced by the DMA_UNKNOWN truth value constant node.

In the case of a merged Scope object formed under union combination rules, the ExecuteSearch method of the merged Scope object is permitted to make an internal copy of the original Query Expression, massage it using three valued elimination, and then pass it on to a component scope, so that the ExecuteSearch methods of the component Scope objects see only queries for which

everything is defined as far as the metadata of the component Scope object is concerned.

3.5.7.1.4.3 Operations Across Document Spaces

None of the well known operators defined by DMA can be evaluated with operands that come from different component scopes. The merged scope is allowed to treat as undefined any operand which is undefined relative to the component scope to which the query is being delivered. A query expression is evaluated relative to each component scope as if the operands from other component scopes are undefined. Three valued elimination is used to eliminate the undefined operands in the query.

A Document Space may or may not support three valued elimination, but it is required to support three valued logic. If three valued elimination is needed and is not available, the query fails with a query construction error.

3.5.7.1.5 Orderings

The Orderings property is a list of Query Order By Node objects that control the ordering of the Query Result Rows. Each Query Order By Node object refers to an element in the Selections list via its Selections Index property, and indicates whether the order is ascending or descending via its Descending Requested property. The ordering is a multi-part ordering. The main ordering is on the first element in the Orderings list. Then, the ordering within that is on the second element in the Orderings list, etc.

The collating sequence to be used by all the Scope objects involved in the query can be specified by setting the Collation Sequence Id property of the Query object. If this property is not set in the Query object, then (a) if the Scope is for a Document Space, its default sequence is used, and (b) if the Scope object is a merged Scope object, an attempt is made to find a collation sequence supported by all the component scopes. If the component scopes do not have a collation sequence in common, the ExecuteSearch method returns an error.

Properties that are defined in the current Query Result Row may have a NULL value. As far as the Orderings list is concerned, the NULL value collates before all other values.

3.5.7.1.6 Distinct Rows Requested

The Distinct Rows Requested property of the Query Root object specifies whether duplicate Query Result Rows are to be discarded or not. If duplicates are to be discarded, you can expect a sort to be involved, and you can expect to wait until all Query Result Rows are retrieved and sorted before the first Query Result Row will be returned by the IdmaResultSet:: GetNextResultRow method on the Query Result Set object.

If the Has Arbitrary Order By property is DMA_TRUE for all component Scope objects of a merged Scope object, then the middleware (i.e., the software implementing the merged Scope object) need not perform a sort and suppress duplicate Query Result Rows. What the middleware is free to do instead is to extend the Orderings list in the copies of the query passed to the component Scope objects such that it includes all the elements of the Selections list.

Then, the middleware can perform a merge on the Query Result Row objects returned from the component Query Result Set objects instead of a sort. Note that the middleware can't just set the Distinct Rows Requested to DMA_TRUE in the copies of the Query object passed on to the component Scope objects and assume there will be no duplicate Query Result Rows. While doing that would cause the component Query Result Set objects to return distinct Query Result Rows, there can still be duplicate Query Result Rows across the component Query Result Set objects, and it is not necessarily the case that the Query Result Rows returned by the component Query Result Set objects will all be sorted in exactly the same way.

3.5.7.2 Example Query Object

Suppose one wanted to find the titles of all documents with "Smith" as any part of the name of the Author. This could be done with the query "SELECT Title FROM DocVersion(InclSubclasses = TRUE) WHERE Author LIKE '%Smith%'". This would find all documents regardless of their searchable class. The Query object and its dependent subobjects would look like the following:

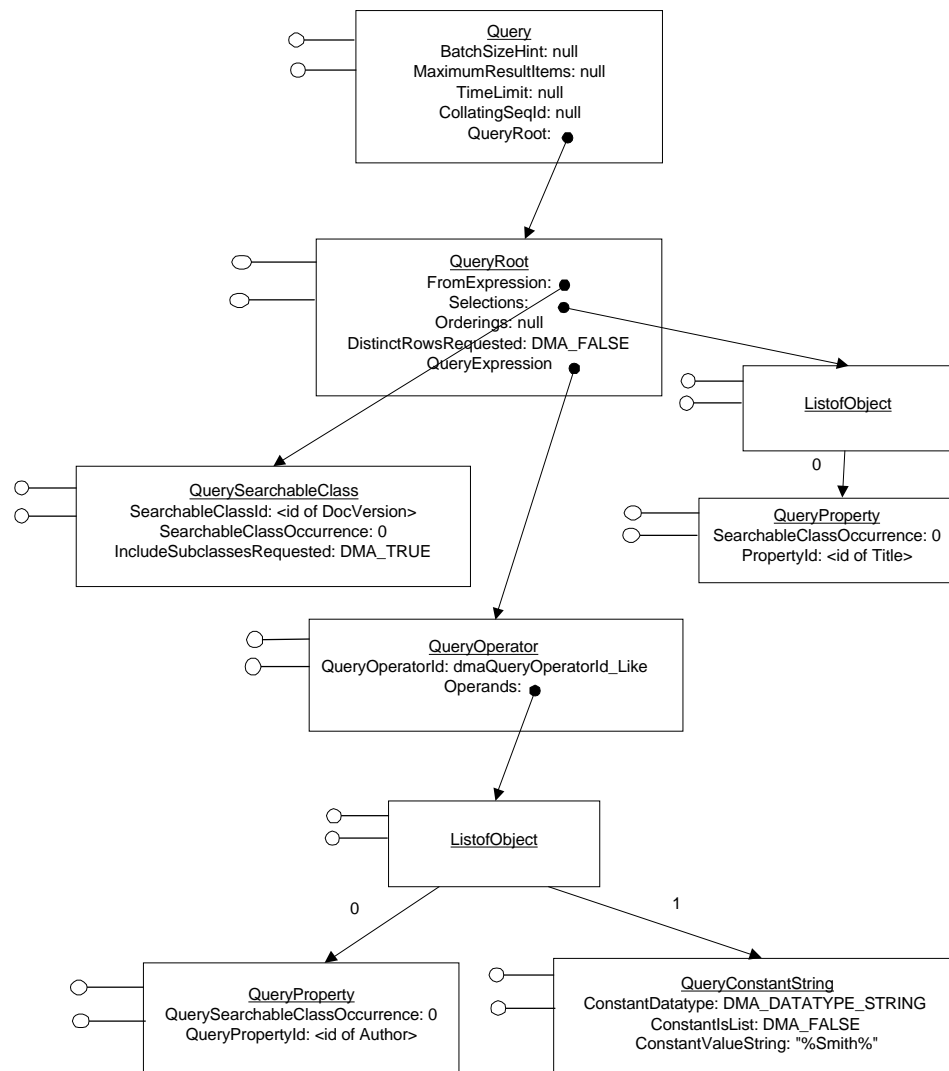


Figure 3-11: Example Query Object

3.5.8 Query Result Sets

A Query Result Set object is the vehicle for delivering the results produced by a query. The objects returned by calling `IdmaResultSet:: GetNextResultRow` of a Query Result Set object are called Query Result Row objects. They are delivered in the order specified in the Orderings list of the Query object, assuming that the Ordering list is not null. Each element of the Orderings list refers back to an element in the Selections list of the Query object.

3.5.8.1 Query Result Set Objects

Query Result Set objects are returned by `IdmaScope:: ExecuteSearch`.

All Query Result Set objects support the `IdmaResultSet` interface, which includes the methods `GetNextResultRow`, `TerminateResults`, and `ReExecuteQuery`.

The `GetNextResultRow` method is used to produce each Query Result Row object that satisfies the query.

The `TerminateResults` method stops further query results from being produced from the underlying Query Result Set Object.

The `ReExecuteQuery` method can be used to restart the query without rebuilding the Query object and without calling `ExecuteSearch` again. The Query Result Row objects produced (by `GetNextResultRow`) after `ReExecuteQuery` is called will reflect any relevant changes made to the persistent stores of the document spaces involved in the interim.

3.5.8.2 Query Result Row Objects

Query Result Row objects are generated by `IdmaResultSet::GetNextResultRow`. The result row object contains a sequence of properties that correspond one-to-one with the properties specified in the Selections list of the query.

Unlike the other DMA objects, the Property Description list of the Class Description object of a Query Result Row object must be created dynamically at query execution time. This is because the Selections list elements are not known until `IdmaScope::ExecuteSearch` is called to generate a Query Result Set object.

The Query Result Row object is self describing, and every Query Result Row produced from the same Query Result Set object for the same query has the same metadata information. The Property Descriptions list of the Class Description object of a Query Result Row object begins with Property Descriptions of some properties defined by DMA (and possibly some implementation defined properties), and ends with a contiguous list of the Property Descriptions of the Properties in the Selections list, in the same order in which these Properties appear in the Selections list.

The value of the Select List Offset property is an index into the Property Descriptions list of the Class Description of the Query Result Row object of which the Select List Offset property is a Property. The value of the Select List Offset property is the list index of the Property Description of the first Property in the Selections list.

Thus, given a Query Result Row object, the client can obtain the value of the Property Descriptions list property (say *L*) of its Class Description, as well as the integer value of its Select List Offset property (say *X*). Then, starting with the list element at index *X* of list *L*, the client can sequence through the Property Descriptions of the Properties in the Selections list, in order of increasing

list index. Similarly, by calling the appropriate `IdmaProperties:: GetPropVal{datatype}ByIndex` method with index value equal to X plus the (zero relative) index into the Selections list, the client can obtain the values corresponding to the Selections list elements in the current Query Result Row object. This is true, even if the same property appears more than once in the Selections list. In contrast, note that the `IdmaProperties:: GetPropVal{datatype}ById` methods can not be used to get the values of all the Selections list elements if a property is selected more than once, because these methods have no way to distinguish between the multiple occurrences.

Therefore, using the `GetPropVal{datatype}ByIndex` methods instead of the `GetPropVal{datatype}ById` methods of `IdmaProperties` to get the values of the properties of a Query Result Row that were in the Selections list of the query is required, so that Query Result Row properties can keep their proper Id values, even if they appear more than once in the Selections list. The `GetPropVal{datatype}byId` and `PutPropVal{datatype}ById` methods return the error `DMARC_BAD_PROPID` when used on Query Result Row objects to access properties that appear in the Selections list of the query. This restriction does not apply to properties in the Query Result Row that were not in the Selections list of the query.

Each Query Result Row object produced from the same Query Result Set object has the same metadata information. This is true, even if some selected elements have null values in the current Query Result Row object. If an attempt is made to obtain the value of a property from the current Query Result Row object that is null, the `DMARC_VALUE_NOT_SET` result code is returned (the same as for any property of any DMA object that has a null value).

Query Result Row objects do not support the `IdmaConnection` interface.

However, if the Object Instance Id property is in the Selections list, its value can be obtained and passed to the `IdmaDocSpace:: ConnectObject()` method (assuming this method is available). The `ConnectObject` method will then attempt to create a scratchpad DMA object connected to a persistent object in the target document space.

Likewise, if a Selections list element is an object valued property (such as the This property of a class in the From list), and the element has a corresponding value in the current Query Result Row, accessing that object valued property will result in a DMA object that provides the appropriate interfaces. Thus, for example, the object might provide the `IdmaConnection` interface.

Query Result Row objects delivered to the client must be released by the client.

3.5.8.2.1 Result Row Class Descriptions

3.5.8.2.1.1 Duplicate Id's

It is not possible to search properly within a single document space if two classes have the same Id. Therefore, this is considered a bug in the document space schema. Similarly, properties should only have the same Id (i.e., has an alias in common) if they are the same. It is a rule of the metadata model that the property descriptions of a single class description be distinct – they have no property Id in common.

This rule is relaxed for Query Result Row Class Descriptions. They are synthetic, and dynamically created depending upon the query. In constructing the Query Result Row Class Description it is permissible to reuse the corresponding property Id's and property descriptions of Selections elements, even though this may result in duplicate properties in the Result Row Class Description. The Result Row properties corresponding to the Selections list properties may be only by accessed by index, not by Id, as explained above. The only properties of the Result Row accessible by Id are those not produced from the Selections list. This prevents duplication of Id's having any effect on the use of the result row, including access to properties of the Result Row object itself.

The client can construct a query with the same property in the select list more than once. This probably won't matter, unless, possibly, you want to save the Query Result Set object and query it later according to Properties in its Selections list. Therefore, it is allowed but not required that an implementation disambiguate the properties in a Query Result Set Class Description.

3.5.8.2.1.2 Synthesized Object Properties

A Query Result Set object may, but is not required to, synthesize Query Result Row class property descriptions for its projection of Selections list elements. If property descriptions are synthesized, rather than reused from some other metadata space, there are a number of properties in the Selections property descriptions that become irrelevant and are appropriately omitted in the Query Result Row class description.

The following table illustrates allowable simplifications of the Property Description properties in the synthesized Query Result Row Class Description, for those properties in the Selections list of the query. All synthesized Query Result Row properties are effectively System Generated and Read Only, whether or not the Property description specifies that.

Table 3-1 Property Descriptions

Property Name	Implemented	Value Req'd	Comment
OIID	-	-	Not useful
Class Description	Yes	Yes	Some metadata space's property Description Class applicable to this Property Description instance
This	-	-	Not useful
Create Pending	-	-	Not useful
Update Pending	-	-	Not useful
Delete Pending	-	-	Not useful
Display Name	Yes	-	optional: ideally the scope's name for the selected property, perhaps with some qualification that reflects the selection or From class.
Descriptive Text	Yes	-	optional: ideally the scope's description for the selected property, reflecting the nature of the selection also.
Ids	Yes	Yes	Not useful. Can be the scope's values.
Property Data Type	Yes	Yes	The correct data type (e.g., object)
Cardinality	Yes	Yes	As appropriate.
Is Selectable	Yes	Yes	Not useful.
Is Searchable	Yes	Yes	Not useful.
Is Orderable	Yes	Yes	Not useful.
Query Operator Descriptions	Yes	-	optional: Empty list preferred to maintain metadata space simplicity.
Is System Generated	Yes	Yes	Always true.
Read Only	Yes	Yes	Always true.
Is Value Required	Yes	Yes	true only if values were required for the selected property of the scope and all result rows have a value for the property.
Is Hidden	Yes	Yes	Always false for result row properties corresponding to Selections elements; can be true for all other properties.
Default Value	Yes	-	Not useful.
Property Selections {data type}	Yes	-	Not too useful. Can be hints to application about values that will be found. Copy the values from the scope's Property Description. For object valued properties, should be omitted to simplify metadata space interactions.

Property Name	Implemented	Value Req'd	Comment
Property Maximum {data type}	Yes	-	Can be hint to applications about the range the values will be in. Obtained from the scope's Property Description.
Property Minimum {data type}	Yes	-	Can be hint to applications about the range the values will be in. Obtained from the scope's Property Description.
Required Class	Yes	-	Not useful. Null value maintains metadata space simplicity for the result set.
Reflective Property Id	Yes	-	Cannot be synthesized. Should always be omitted.

3.5.9 Searching Across Multiple Repositories

The key feature of DMA is the ability to perform queries across multiple, heterogeneous, legacy Document Space repositories. Clients accomplish this by calling the `IdmaScope::ExecuteSearch` method on a merged Scope object and then using the merged Query Result Set object returned by `ExecuteSearch`. By using the merged Scope object and merged Query Result Set object, the client is, in effect, able to treat the combination of the component Scope objects as if they were a single Scope object. The software that implements the merged Scope object, the merged Query Result Set object, merged Query Result Row objects and that coordinates the queries across the component Scope objects is called middleware.

The merged Scope object (1) presents a unified view of the set of Document Space (and/or merged Scope) objects that are its Component Scope objects, (2) provides coordination of queries across its Component Scope objects, and (3) merges Query Result Row objects from the Query Result Set objects produced by its Component Scope objects into a single sequence of Query Result Row objects.

Merged Scope objects, the merged Query Result Set objects, and the merged Query Result Row objects present exactly the same set of DMA COM interfaces as do Scope objects, Query Result Set objects, and Query Result Row objects of individual Document Spaces, respectively.

It is required that the middleware rely on only the public DMA interfaces on its component Scope objects to merge their metadata into a single set of metadata for the merged Scope object. It is also required that the middleware rely on only the public DMA interfaces of its component Scope objects, the Query Result Set objects, and the Query Row objects to perform queries. Since only publicly available interfaces are used, any client could provide the functionality of merged scopes, if it so desired. The middleware may take advantage of additional interfaces and additional properties of the component Scope and Result Set objects, e.g., for performance reasons, but it must not depend on any such provisions in order to operate successfully.

3.5.9.1 Creating a Merged Scope Object

The following object instance diagram illustrates the creation of a merged Scope object from component Scope objects.

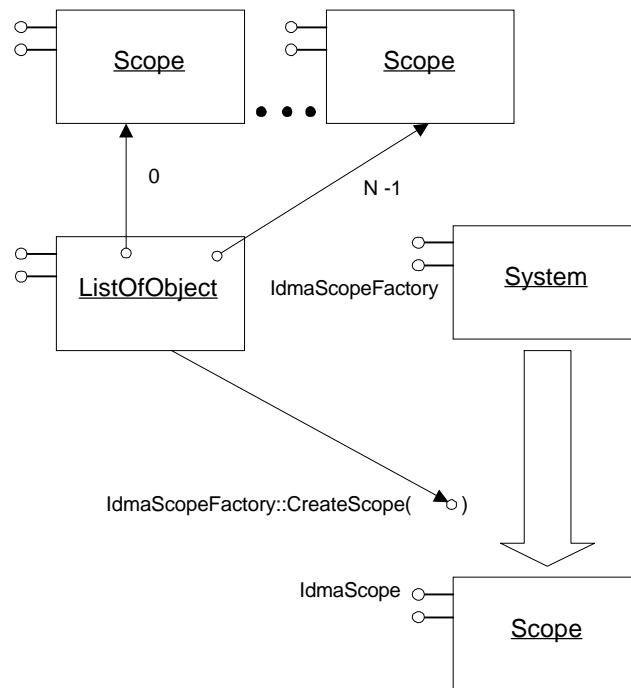


Figure 3-12: Creation of a Merged Scope Object

A Merged Scope object is constructed from a list of component Scope objects (typically these would be Document Space Scope objects). The middleware merges the metadata of the component Scope objects and presents a unified view of the merged metadata as the metadata of the merged Scope object.

This specification does not provide for coordinated update across multiple repositories. Update of only one document space repository at a time is specified.

This specification does not provide for updating of metadata.

The only mechanism specified for cross repository references (including containment) is that a transfer element of a rendition of a document may be a URL.

The merged Scope object is constructed from other Scope objects. Therefore the operations that it can perform are limited by the methods that can be performed on its component Scope objects. Thus, if one of its component Scope objects loses its Document Space connection, the ExecuteSearch method will become unavailable on that component Scope object.

Since a Merged Scope object presents a merged view of the metadata of several component Scopes that can be very different, there is a distinct possibility that the merged Scope object is not very useful. However, in general, given that there is a reasonable attempt by Document Space administrators to use common classes of objects and properties, merged Scope objects can be a powerful tool.

If a merged Scope object is built using the union rules (see below), a client may be able to construct a query expression that a Document Space may not fully understand. If this is the case, then three valued elimination (see the above section on “Partially Defined Query Expressions”) can optionally be used to make the query meaningful. The `bRequestElimination` parameter to the `ExecuteSearch` method influences how three valued elimination is used to reduce partially understood queries to fully understood queries. Clients (other than merged Scope implementations) should pass `DMA_FALSE` for the value of this input parameter, for then `ExecuteSearch` will return errors for malformed queries instead of attempting to execute the reduced query and possibly returning confusing results. Returning errors is normally the desired behavior in this case.

3.5.9.2 Merged Scopes Are an Optional Feature

In order to support the trivial case of the use of DMA without merged Scope objects, the existence of merged Scope objects is optional. That implies that the System object is not required to support the `IdmaScopeFactory::CreateScope` method. In the event that this method is not supported, the client is limited to performing query operations to a single Document Space, or must perform its own unification of Document Space Scopes.

3.5.9.3 Query Execution With Merged Scopes

The following diagram illustrates the objects involved in executing a query on a merged Scope object.

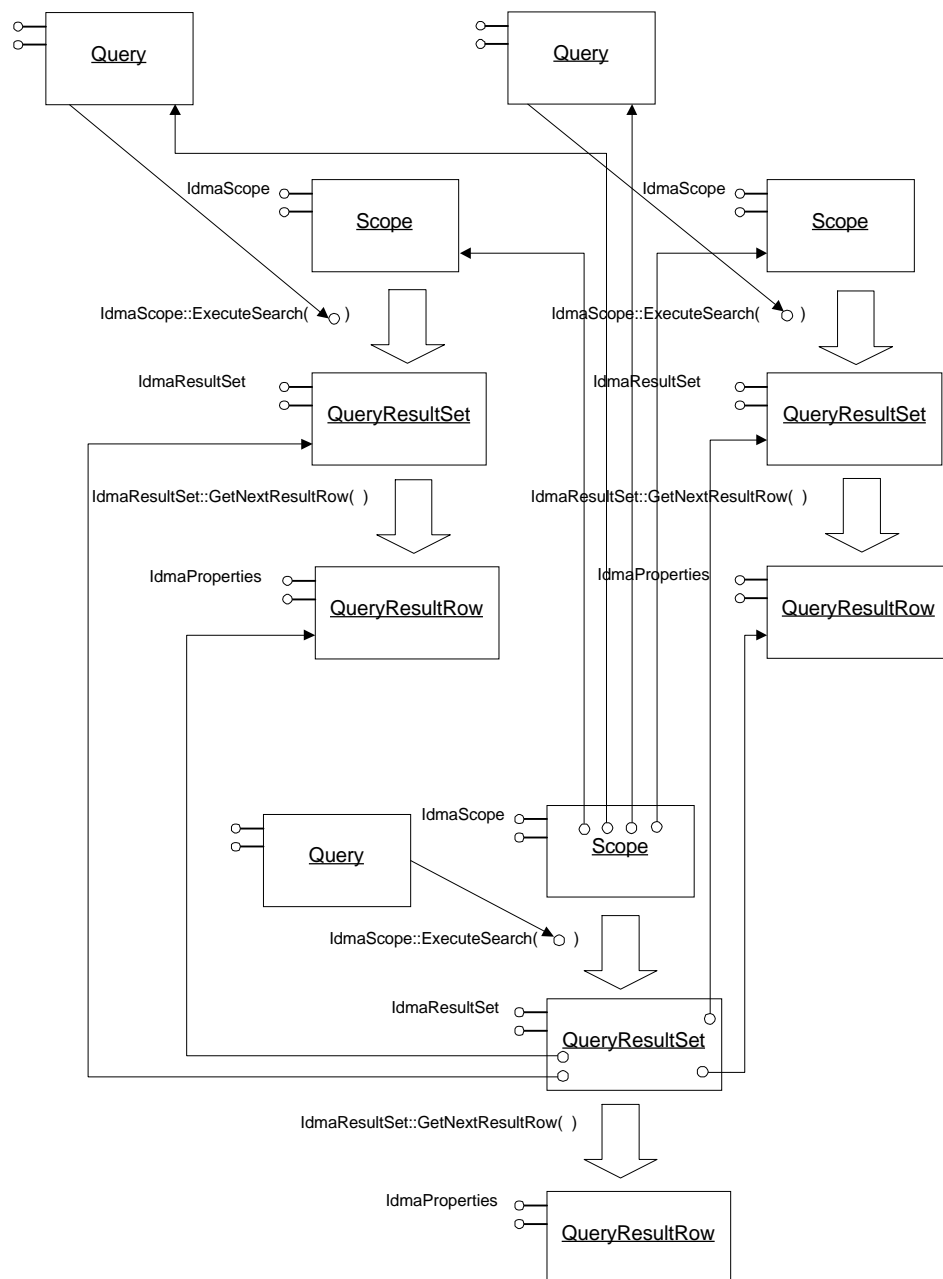


Figure 3-13: Objects Involved With Merged Scope

In the above diagram, the lowest Scope object in the diagram is the merged Scope object. It keeps internal interface pointers to two component Scope objects which are higher in the diagram. Internal interface pointers are indicated by thin line arrows in the diagram. Method invocation on one object that generates a new object is indicated by a wide arrow that is labeled with the name of the method invoked. The client's original Query object is the lowest Query object in the diagram. The client's call on the ExecuteSearch method takes this object as a parameter. This is indicated by a thin line arrow from the Query ob-

ject to the parentheses of the ExecuteSearch method. The upper two Query objects in the diagram are generated by the client's call on ExecuteSearch. The two generated Query objects need not be identical to the original Query object. The middleware is allowed (but not required) to copy the original Query object, massage it (e.g., by using three valued elimination, or by modifying the Orderings list, etc.), and pass it on to the ExecuteSearch method of a component Scope object. Remember that ExecuteSearch does not keep a pointer to the Query object passed to it. Therefore, the upper two Query objects in the diagram would be released by the middleware after it calls the ExecuteSearch method on the component Scope objects. The Query Result Set object generated by the client's original ExecuteSearch call keeps internal interface pointers to the Query Result Set objects generated by the secondary ExecuteSearch calls on the component Scope Objects.

The Class Description of the Query Result Row objects are synthesized from the Selections list and the metadata of the underlying Scope object by ExecuteSearch. The Property Description list of this Class Description includes descriptions of properties corresponding to all elements of the Selections list. There is a positional correspondence between the elements of the Selections list of the Query object and a sequence of Property Descriptions in the Property Description list of the Query Result Row object. This has already been described above.

If the Orderings list of the Query object is not null, Query Result Row objects are enumerated by the IdmaResultSet:: GetNextResultRow method of the merged Query Result Set object in the order defined by the Orderings list. As stated above, the middleware might enhance the Orderings list. A reason the middleware might do this is to return unique Query Result Row objects by performing a merge instead of a sort of the Query Result Row objects returned by the component Query Result Set objects.

The ExecuteSearch method on a merged Scope object is required to call the ExecuteSearch method on each of its component Scope objects before it returns.

3.5.9.4 Merging Metadata

If there were no commonality across document spaces, then querying across multiple repositories would be uninteresting. But, as it turns out, there usually is, in fact, some commonality across document spaces. However, this commonality is often obscured by technical details.

For example, the name of a subclass of DocVersion might be "Depositions" in one document space, and "depositions" in another, and "DEPOS" in a third. A fourth document space might have the name of the depositions class in Kanji or Shift-JIS or Hebrew character set. And yet, these searchable classes are semantically close enough, that the client wishes to consider them the same for purposes of his query. This same problem exists for properties and rela-

tionship types. The problem may be referred to as the **unification** problem. We wish to unify these different searchable classes (and properties and relationships) across the document spaces when they are semantically the same as far as the users are concerned. DMA abandons the approach of unifying searchable classes, properties, relationship types, etc. by name. Instead, DMA unifies strictly by Id.

3.5.9.4.1 Id's

Searchable class Id's, property Id's, Query Result Set Id's, Query operator Id's, collating sequence Id's, etc. are of type Dmald, which is defined to be a standard DCE UUID structure. DCE UUID's are unique over all space and time. COM clients are familiar with DCE UUID's, because COM interface ID's are DCE UUID's. Reliance is placed on the wonderful feature of Dmald values that each Dmald value ever generated is unique over all time and space. This prevents naming collisions without the need for a central registration authority, and thereby facilitates adding properties, adding operators, add collating sequences, uniquely identifying queries in progress for purposes of cancellation, etc.

Searchable classes, properties, relationship types, etc. are assigned Id's to identify them. In order to promote cross repository query interoperability, each searchable class, property, and relationship type can potentially be assigned a list of alias Id's. DMA defines two properties (or classes) to be semantically the same if the intersection of the Ids property lists is non-null, i.e., when the two lists have at least one Id in common. When such a match exists, we say that the properties, (or classes) are unifiable. When such a match does not exist, we say that the properties (or etc.) are not unifiable. Unified searchable classes and properties are considered to be semantically the same for purposes of cross repository query.

When merging metadata either under union or intersection, the resulting value of the Ids property in the unified class or property or operator is always the intersection of the Ids.

The first step in the unification process is merging searchable classes.

3.5.9.4.2 Merging Searchable Classes

A scope object instance includes a snapshot of the metadata of an individual document space, or is a merged Scope, and the metadata is the result of a merge from one or more individual Document Space Scopes. The individual Document Spaces provide Scopes with the metadata of the individual Document Space. The middleware merges component Scope metadata into a merged Scope.

3.5.9.4.2.1 Searchable Class Hierarchy

In general, it is impossible to unify the searchable classes of two component scopes and yet preserve the partial ordering of both class hierarchies in the merged scope. The following are two cases where the partial ordering of the class hierarchies can not be preserved in the merged hierarchy. In the figures, class A unifies with class A', and class B unifies with class B'.

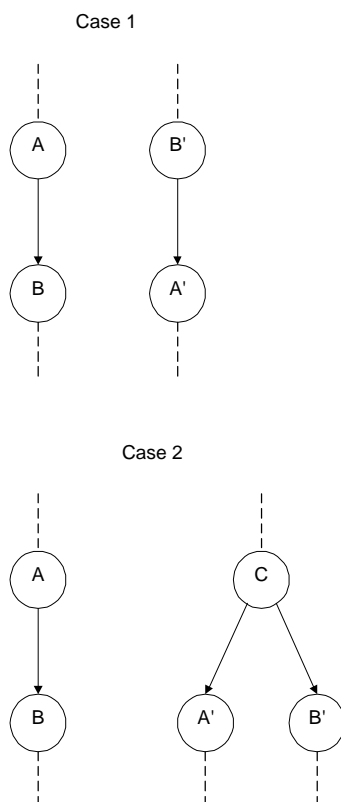


Figure 3-14: Cases Where Partial Ordering Cannot Be Preserved

Therefore, DMA takes the position that a merged scope is permitted (but not required) to have the searchable classes appear as immediate subclasses of class DMA. The Searchable Class Descriptions property lists all the searchable classes of the Scope.

Nothing of critical importance is lost, since the full class hierarchies for all of the individual document space scopes are still available to the client. What is critical is to maximize interoperability. By not requiring strict mapping between the class hierarchies of individual document spaces, interoperability is maximized.

3.5.9.4.3 Merge Options

The client must choose one of two options for merging the metadata of individual document space scopes - union or intersection. The intersection option keeps only the metadata common to all individual scopes. The intersection might be empty. The union option keeps all the metadata of all the individual scopes, unifying as much as it can. It is expected that the union option will be more useful to DMA clients. It is also expected that the query expression part of most queries will concentrate mostly on unifiable searchable classes and properties.

In DMA 1.0 neither union nor intersection of metadata is required to be supported.

The creation of a merged scope is allowed to fail for certain combinations of component scopes under both union and intersection.

The searchable classes of the component scopes are merged in a pairwise manner. Thus, if class A in the first component scope unifies with class A' in the second component scope, that is because they have at least one alias class ID in common. If class A'' in the third component scope unifies with class A' in the second component scope, that is because they have at least one alias Id in common. However, there is no guarantee that class A and class A'' have a common alias Id. We take the position that the third class does not unify with the first two. Thus, the final result of unification may depend upon the order in which unification is attempted. In order to make the result definite, we specify the following rules.

- 1 The component scopes are scanned in increasing list index order. The final merged scope is developed incrementally by merging in one component scope at a time.
- 2 The classes of the current merged scope and the current component scope are scanned in increasing list index order. Each class in the component scope is attempted to be merged with the current class in the merged scope. The outer loop is on the merged scope, and the inner loop is on the component scope. When two classes unify, the alias list of the final merged class is the intersection of the two alias lists, both under union and intersection rules.
- 3 When two classes unify, the properties of the classes are scanned in increasing list index order. The outer loop is on the class in the merged scope, and the inner loop is on the class in the component scope. When two properties of two classes unify, the alias list of the final merged property is the intersection of the two alias lists, both under union and intersection rules.

Once it has been discovered that two searchable classes unify, their metadata must be unified under the option chosen - union or intersection.

3.5.9.4.4 Merging Class Descriptions

3.5.9.4.4.1 Merging the Name Property Index Property

The Name Property Index property of a merged class description will have a value if and only if all the component classes which unified into the merged class have a value for Name Property Index and the properties thereby identified unify.

3.5.9.4.4.2 Merging Property Descriptions

If two properties should be unified according to their primary and alias Id's, then they both must have the same base datatype. Also, the two properties must also have the same cardinality. In other words, both must be scalars, or both must be lists, or both must be enumerations. Otherwise, the merge fails and returns an error. The sizes of the two properties may differ, their maximum and minimum values may differ, and their list of allowable values may differ.

The following are the rules for combining the Property Descriptions of two unified searchable classes under the union option:

- Property Descriptions are unified on the basis of their Ids property. The resulting value of the Ids property is the intersection of the two Ids properties.

If a Property Description for the new searchable class does not unify, it is added to the list of Property Descriptions being developed.

If a Property Description for the new searchable class unifies, the Property Description developed so far is modified according to the following rules:

- Compare the descriptive properties (e.g., property name, descriptive text). Use the properties of the first Property Description.
- Compare the constraint properties of the two Property Descriptions. The merge should weaken the constraints. That means select the maximum of the two "max value" properties, and select the minimum of the two "min value" properties. If there is a value constraint list, then union the values.
- Compare the Default Value {data type} property of the two Property Descriptions. If both have a non null value, and if these values are equal, then this value is the value for the Default Value {data type} property. Otherwise, there is no value for the Default Value {data type} property.
- Compare the Is Selectable properties of each property description. If one property is selectable and the other is not, then make the Is Selectable property of the synthesized Property Description be selectable.

- Compare the Is Searchable properties of each property description. If one property is searchable and the other is not then make the Is Searchable property of the synthesized Property Description be searchable.
- Under union merge, the Is Orderable property should set to DMA_TRUE if the merged scope implements sorting of the component scope result sets. Otherwise, it is formed from the AND of the Is Orderable property values of the component scopes.. However, if the merged scope has no supported collation sequences and chooses not to fail scope creation, the Is Orderable property of any string property description should be set to DMA_FALSE.
- The Required Class property of a merged property description will have a value if and only if the required classes of all the component properties unify.
- The Reflective Property Id property of a merged property description will have a value if and only if the required classes of all the component properties unify, and every component property has a reflective property id, and all the properties thereby identified unify.
- The value of the Is Value Required property is DMA_TRUE in the merged scope if and only if it has the value DMA_TRUE in all the component scopes, i.e., it is the AND of the values in the component scopes.

The following are the rules for combining the Property Descriptions of two unified searchable classes under the intersection option:

- Property Descriptions are unified on the basis of their Ids property. The resulting value of the Ids property is the intersection of the two Ids properties.

If a Property Description for the new searchable class does not unify, it is discarded.

If a Property Description for the new searchable class unifies, the Property Description developed so far is modified according to the following rules.

- Compare the descriptive properties (i.e., property name, descriptive text). Keep the properties of the first Property Description.
- Compare the constraint properties of the two Property Descriptions. The merge should strengthen the constraints. That means select the minimum of the two “max value” properties, and select the maximum of the two “min value” properties. Intersect the value constraint lists.
- Compare the Default Value {data type} property of the two Property Descriptions. If both have a non null value, and if these values are equal, then this value is the value for the Default Value {data type} property. Otherwise, there is no value for the Default Value {data type} property.

- Compare the Is Selectable properties of each property description. If one property is selectable and the other is not, then make the Is Selectable property of the synthesized Property Description be not selectable.
- Compare the Is Searchable properties of each property description. If one property is searchable and the other is not then make the Searchable property of the synthesized Property Description be not searchable.
- Under intersection merge, the Is Orderable property is formed from the AND of the Is Orderable property values of the component scopes. However, if the merged scope has no supported collation sequences and chooses not to fail scope creation, the Is Orderable property of any string property description should be set to DMA_FALSE.
- The Required Class property of a merged property description will have a value if and only if the required classes of all the component properties unify.
- The Reflective Property Id property of a merged property description will have a value if and only if the required classes of all the component properties unify, and every component property has a reflective property id, and all the properties thereby identified unify.
- The value of the Is Value Required property is DMA_TRUE in the merged scope if and only if it has the value DMA_TRUE in all the component scopes, i.e., it is the AND of the values in the component scopes.

3.5.9.4.5 Merging Scope Properties

3.5.9.4.5.1 Merging the Arbitrary Order By Property

The scope property Has Arbitrary Order By should be set to DMA_TRUE if the merged scope implements sorting of component scope result sets. Otherwise, it should be set to the AND of the values of the Has Arbitrary Order By property of the component scopes.

3.5.9.4.5.2 Merging the Distinct Property

The scope property Has Distinct should be set to DMA_TRUE if and only if the merged scope supports returning only distinct result rows. This will depend on the abilities of the merged scope implementation as well as the characteristics of the component scopes as far as their values for the Has Arbitrary Order By and Has Distinct properties.

3.5.9.4.5.3 Merging the Collation Sequence Ids Property

The Collation Sequence Ids list property of the merged scope is generated as follows:

- First, the intersection of the lists from the component scopes is formed. If the resulting list is empty, merged scope creation will fail with error DMARC_NO_COLLATIONS.
- Then, any id which specifies a collation sequence which is neither implemented natively by the merged scope nor supported by an installed text ordering service is removed. If the result of this processing is an empty list, merged scope creation may fail (with DMARC_NO_COLLATIONS), or alternatively adjustments may be made to the Is Orderable property of any string property descriptions, as noted above.

3.5.9.4.5.4 Merging the Operators Property

Merging the list of query operators is straightforward. For intersection, only the operators common to all child scopes survive. For union, they all survive.

Both query operators and join operators are in the list of operators of the scope object.

3.5.9.4.5.5 Merging Query Construction Classes

The properties of the well known query construction classes in the merged scope must include at least the union of the properties of the corresponding classes from the component scopes, whether merging under intersection or union rules.

The merged scope must present the Query Construction Classes in a class hierarchy which conforms to that of the DMA specification.

3.5.9.4.6 Merging Query Operand Descriptions

3.5.9.4.6.1 Merging the Operand Data Type Property

For intersection merge, either the values of the two Operand Data Type properties must be the same, or one of the values must be DMA_DATATYPE_ANY, in which case the value of the merged property is the value of the other property.

For union merge, either the values of the two Operand Data Type properties must be the same, or one of the values must be DMA_DATATYPE_ANY, in which case the value of the merged property is DMA_DATATYPE_ANY.

3.5.9.4.6.2 Merging the Boolean Properties

In general, the Boolean properties (Allows Singleton, Allows List, Allows Constant, Allows Property, Allows Expression) are AND'ed for intersection, and OR'ed for union merge.

3.5.9.4.7 Merging Query Operator Descriptions

3.5.9.4.7.1 Merging the Result Type Property

For the merge to succeed, the value of the two Result Type properties must be equal, and must indicate a base data type (i.e., must be one of DMA_DATATYPE_BINARY, DMA_DATATYPE_BOOLEAN, DMA_DATATYPE_DATETIME, DMA_DATATYPE_FLOAT64, DMA_DATATYPE_ID, DMA_DATATYPE_INTEGER32, DMA_DATATYPE_OBJECT, DMA_DATATYPE_STRING). The merged value is the same as the value of the two properties.

3.5.9.4.7.2 Merging the Is List Property

For the merge to succeed, the value of the two Is List properties must be the same. The merged value is the same as the value of the two properties.

3.5.9.4.7.3 Merging the Is Safe To Eliminate Property

The Is Safe To Eliminate property is AND'ed under both the intersection and union options.

3.5.9.4.7.4 Merging the Join Participation Property

For intersection merge, the following rules are considered in sequence until one is applicable:

- 1** If the value of the Join Participation property of either class is not equal to DMA_JOIN_PARTICIPATION_NONE, DMA_JOIN_PARTICIPATION_OPERAND, or DMA_JOIN_PARTICIPATION_OPERATOR, then the merge fails with a DMARC_OPERATOR_MERGE_CONFLICT error.
- 2** Otherwise, if the value of the Join Participation property is equal for both classes, then the value of the merged property is the same as that of the two classes.
- 3** Otherwise, if the value of the Join Participation property of one class is DMA_JOIN_PARTICIPATION_NONE, and the value for the other class is DMA_JOIN_PARTICIPATION_OPERAND, then the value for the merged property is DMA_JOIN_PARTICIPATION_NONE.
- 4** Otherwise, the value of Join Participation is DMA_JOIN_PARTICIPATION_OPERATOR for one class, and a different value for the other class, and the error DMARC_OPERATOR_MERGE_CONFLICT is returned for the merge operation.

For union merge, the following rules are considered in sequence until one is applicable:

- 1** If the value of the Join Participation property of either class is not equal to DMA_JOIN_PARTICIPATION_NONE, DMA_JOIN_PARTICIPATION_

OPERAND, or DMA_JOIN_PARTICIPATION_OPERATOR, then the merge fails with a DMARC_OPERATOR_MERGE_CONFLICT error.

- 2 Otherwise, if the value of the Join Participation property is equal for both classes, then the value of the merged property is the same as that of the two classes.
- 3 Otherwise, if the value of the Join Participation property of one class is DMA_JOIN_PARTICIPATION_NONE, and the value for the other class is DMA_JOIN_PARTICIPATION_OPERATOR, then the value for the merged property is DMA_JOIN_PARTICIPATION_OPERATOR.
- 4 Otherwise, the value of Join Participation is DMA_JOIN_PARTICIPATION_OPERATOR for one class, and the other class has a different value, and the error DMARC_OPERATOR_MERGE_CONFLICT is returned for the merge operation.

3.5.9.4.7.5 Miscellaneous Merging Rules

Usually, but not always, Boolean flags are AND'ed for intersection, and OR'ed for union.

The merged scope is required to construct a metadata space which conforms to the object model specification and includes the query construction classes and the unified searchable classes merged according to the merge rules.

3.6 DMA Containment Model

3.6.1 Introduction

Webster's Ninth New Collegiate Dictionary defines containment as:

"The act, process, or means of containing"

Contain is defined as:

"To have within"

Frequently, containment is referred to as "foldering" in document management and imaging systems, meaning the act of storing documents or images within folders as a means of organization. Within DMA, we have chosen to use the word container as a more neutral, canonical term, describing the semantics of objects containing other objects.

This DMA containment model is intended to be sufficiently general and expressive to allow DMA providers to support a number of specialized containment metaphors, such as libraries, electronic file cabinets, electronic file drawers, folders, and so on. However, it is beyond the scope of this version of the DMA specification to define a well-known class that has both containment and content properties. Support for such objects may be added in a future version of the DMA specification, when support for compound documents is added.

3.6.2 DMA Containment Model Overview

The DMA containment model can be characterized as follows:

- Containment is optional. DMA document spaces may choose the following options:
 - Support no containment at all
 - Support classes that provide containment
- Containment is expressed through three mechanisms
 - Navigating containment hierarchies through traversal of properties using the `IdmaProperties` interface.
 - Expressing queries that include containment properties.
 - Updating containers to insert and remove children and containees.
- Two types of containment are defined: direct and referential.
 - *Direct containment.* This models a 1:N, or one-to-many, relationship. A containing object may contain multiple objects, but an object is directly contained within at most one containing object. A containing object that directly contains another object is the *parent* of that object. An object that is directly contained within a parent object is a *child* of that parent. Multiple directly contained objects are *children*. Direct containment defines a strict hierarchy of objects with no cycles. Document spaces implementing direct containment must prevent cycles. For example, moving a parent into one of its children must be disallowed. DMA does not limit the depth of the direct containment hierarchy, but a document space may choose to impose a limit.
 - *Referential containment.* This models an N:M, or many-to-many, relationship. Objects that are referentially contained within a containing object are referred to as *containees*. A containing object that referentially contains another object is referred to as a *container* of that object. A container may contain multiple containees. A containee may be contained within multiple containers. Cycles in referential containment relationships are not disallowed by DMA, because they could be expensive to prevent. A document space may choose to disallow them, however, in which case it is obligated to enforce this. DMA defines operations that modify containment hierarchies in such a fashion that cyclical referential containment can be supported.
- If a document space chooses to support containment, it may support direct containment, referential containment, or both.

- The relationship between a container and a contained object is modeled explicitly in DMA with the introduction of a third object, the ContainmentRelationship object, and its subclasses DirectContainmentRelationship and ReferentialContainmentRelationship. These objects provide a home for “edge data”, data which can be thought of as belonging to the edge between two vertices, the containing object and the contained object.
- Containment is expressed as a set of capabilities within the architecture. The DMA containment model does not impose a specific implementation on document space service providers.
- Preserving the ordering of containees (or children) of a specific container (or parent) is optional.
- Preserving the ordering of containers of a specific containee is optional.
- Objects in a containment relationship must both reside in the same document space. That is, cross-document space containment is not supported in this version of the DMA containment model, although it may be added in a future version.
- Document spaces may support read-only containers. A read-only container has the following characteristics:
 - Containers may be searched and navigated.
 - Containers cannot be created, modified or deleted.
 - DMA programming interfaces model read-only containers by returning an error for the following methods:
 - IdmaConnection::ExecuteChange (returns DMARC_READ_ONLY)
 - IdmaBatch::ExecuteChanges (returns DMARC_READ_ONLY)
 - IdmaClassDescription::CreateInstance (when requested class is a subclass of dmaClass_ContainmentRelationship, returns DMARC_NOT_CREATABLE)
 - IdmaObjectFactory::CreateObject (when requested class is a subclass of dmaClass_ContainmentRelationship, returns DMARC_NOT_CREATABLE)
 - IdmaEditProperties (on objects of subclass of dmaClass_ContainmentRelationship, returns DMARC_READ_ONLY)
 - IdmaRelationshipOrdering::SetOrdering (returns DMARC_READ_ONLY)

3.6.3 Containment and the DMA Object Model

DMA chooses to express containment capabilities by providing a set of optional classes and properties within the DMA object model, along with a set of containment values in the DocSpaceCapabilities property on the DocSpace object. To discover the overall containment capabilities of a DocSpace, client applications can consult the containment settings in the DocSpaceCapabilities property. Client applications can discover the containment capabilities of any particular class or object by checking for the presence of the optional containment properties on it.

Figure 3-15 (below) shows a subset of the classes in the DMA object model, with an emphasis on containment and versioning. The base class `dmaClass_Containable` introduces properties needed by all DMA objects which can be contained, including properties for the object's direct containment parent and referential containment containers. The classes `dmaClass_DirectContainmentRelationship` and `dmaClass_ReferentialContainmentRelationship` logically tie containers and their contained objects together for direct and referential containment, respectively. (Relationship classes are discussed in more detail below.) Lastly, `dmaClass_Container` introduces properties needed by all DMA objects which act as containers, including properties for children and containee objects.

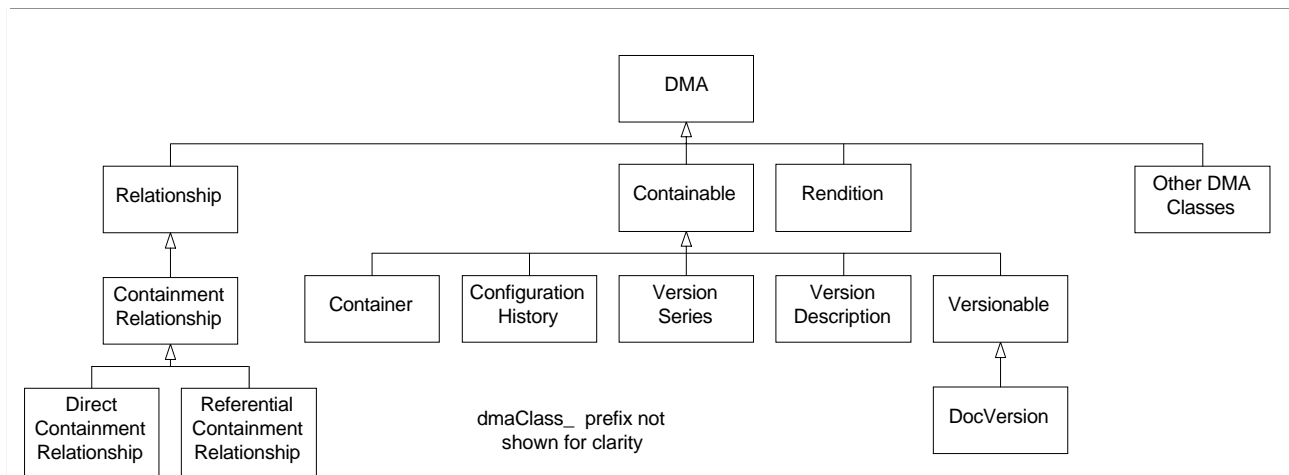


Figure 3-15: DMA Class Inheritance Hierarchy for Containment

Five classes are used to express different types of containment behavior:

Table 3-2 DMA Containment Classes

dmaClass_DocVersion	No containment support. Content supported. Used to model simple documents.
dmaClass_Container	Containment supported. Objects of this class can be used to model containment, but these objects do not directly include content. Useful for modeling folders, libraries, bookshelves, etc.
dmaClass_ContainmentRelationship	Base containment relationship class. If this class is searchable, the document space supports querying for both direct and referential containment relationships in a single query. If not, separate queries are required.
dmaClass_DirectContainmentRelationship	Specialized containment relationship class used to express direct containment only. If this class is supported but not searchable, the document space may support querying for directly contained objects via the dmaProp_ParentContainer shortcut property on the contained object instead.
dmaClass_ReferentialContainmentRelationship	Specialized containment relationship class used to express referential containment only.

A key design goal for DMA is to provide a self-describing system. Applications can determine functionality at run time by interrogating capabilities of document spaces. The DocSpaceCapabilities list on the DocSpace object includes list elements indicating the following characteristics:

- types of containment supported (direct and/or referential, or neither);
- whether containable objects are required to be contained;
- which well-known classes of objects may be contained;
- which ordering options for contained objects are supported;
- maximum depth of containment hierarchy.

An application that connects to a DMA document space may also examine class descriptions to answer the following questions about containment capabilities:

Table 3-3 Interrogating Containment Capabilities

To ask..	Execute..
What classes can I use to create objects that can act as containers?	Check for the presence of well-known containment properties -- <code>dmaProp_Children</code> for direct containment, and/or <code>dmaProp_Containees</code> for referential containment -- in the list of properties for each class description object in the <code>dmaProp_ClassDescriptions</code> property of <code>dmaClass_DocSpace</code> objects.
What classes can I use to create objects that can be contained?	Check for the presence of well-known containment properties -- <code>dmaProp_Parent</code> for direct containment, and/or <code>dmaProp_Containers</code> for referential containment -- in the list of properties for each class description object in the <code>dmaProp_ClassDescriptions</code> property of <code>dmaClass_DocSpace</code> objects.
Can this particular object act as a container?	Check for the presence of well-known containment properties -- <code>dmaProp_Children</code> for direct containment, and/or <code>dmaProp_Containees</code> for referential containment -- in the class description's list of properties for this object.
Can this particular object be contained?	Check for the presence of well-known containment properties -- <code>dmaProp_Parent</code> for direct containment, and/or <code>dmaProp_Containers</code> for referential containment -- in the class description's list of properties for this object.

3.6.3.1 Containment Relationship Objects

The DMA object model associates containers with their contained objects by means of an object of class `dmaClass_DirectContainmentRelationship` or `dmaClass_ReferentialContainmentRelationship`. These objects provide a place to hold “edge data”, that is, data which belong more to the edge between the containing and contained object vertices than to either vertex. Containment relationship objects can also be used to enable arbitrary persistent ordering of the contained objects relative to the containing object, and/or to enable arbitrary persistent ordering of the containing objects relative to the contained object.

Edge data can include both user- and system-defined properties that describe the relationship between the containing and the contained object. For example, the relationship of a folder and a document contained in that folder referentially (Figure 3-13) could have properties such as:

- filer (who filed this document in this folder)
- filing date (when this document was filed in this folder)

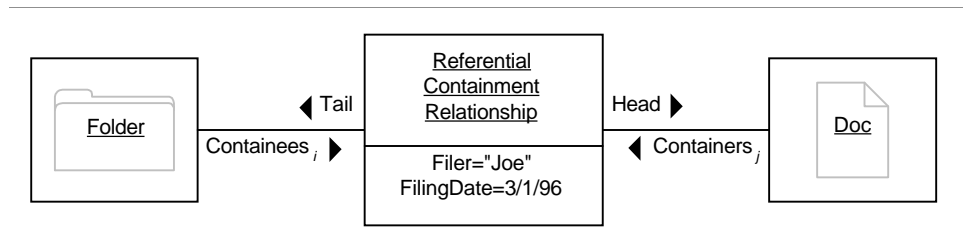


Figure 3-16: Referential Containment Relationship Object Example

These properties logically belong to the relationship between the containing and contained object, rather than to either of the related objects alone. In referential containment, neither the containing nor contained object is a good home for these properties, as the containing object may have many contained objects, and an object may be referentially contained in many containing objects. Each `ReferentialContainmentRelationship` object describes one such containing-to-contained object relationship, providing a natural place to store the properties associated with that single relationship.

For symmetry and consistency of creation and update of containment relationships, both direct containment and referential containment are expressed in the DMA model via relationship objects. However, direct containment is a one-to-many relationship (each child object has only one parent, while a parent can have many children), so it is possible to express direct containment via a property on the child object which directly references the parent, rather than going through an intermediate relationship object. The DMA containment model provides such a property, `ParentContainer` (Figure 3-17), as a shortcut for simpler navigation and query. This property shortcuts the `DirectContainmentRelationship` object and has the parent object as its value. The value is maintained by the document space, hence it is read-only from the perspective of a DMA client. There is no reflective property in the parent object that points directly to the child object, so the shortcut is unidirectional. Updates to direct containment relationships must still be done via the `DirectContainmentRelationship` object. See the “Containment and Query” discussion later in this section for more on this property.

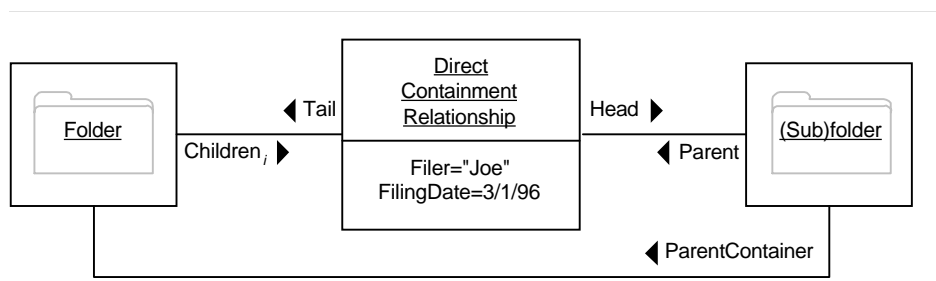


Figure 3-17: Direct Containment Relationship Object Example

3.6.4 Containment and Versioning

Refer to the DMA containment class hierarchy diagram above for the inheritance relationships among DMA classes that describe versioning and containment. This class hierarchy enables objects that model versioning, such as DocVersions, VersionDescriptions, VersionSeries, and ConfigurationHistories, to be contained within a container.

3.6.5 Containment and Persistence

If a DMA document space implements containers, it must provide for persistent storage of containers and their properties. A container is said to be *persistent* when its properties, including its enumeration of the objects contained within it, have been saved to a document space such that the container and its enumeration of objects can be located and accessed by any users of the document space possessing appropriate access rights. Once it is persistent, the container remains accessible until deleted. The sections below describe how containers are created, saved, modified, and deleted; the section is followed by a discussion about maintaining referential integrity in containment relationships.

3.6.5.1 Creating and Saving a Container

A DMA client can create a transient object of class `dmaClass_Container` like any other DMA object by invoking `IdmaClassDescription::CreateInstance()` on the Class Description Object of that class, or by invoking `IdmaObjectFactory::CreateObject()` on the DocSpace object with the appropriate class identifier. The client then uses the `IdmaEditProperties` interface to change the properties of the container. To insert an object into the container using direct containment, the client must first create a transient object of class `dmaClass_DirectContainmentRelationship`. Then the client fills in the properties of the `dmaClass_DirectContainmentRelationship` object, including the Tail (parent) and Head (child) objects. Optionally, the client can set the ordering of the child object within the parent, if the DocSpace supports ordering, by invoking `IdmaRelationshipOrdering::SetOrdering()` on the relationship object.

When the DMA client is ready to save the container (make it persistent), the client first invokes the `IdmaConnection::ExecuteChange()` method on the container and any contained objects in it that are not yet persistent. It can make the parent and child objects persistent in any order, as long as it makes both persistent before calling `IdmaConnection::ExecuteChange()` on the associated `DirectContainmentRelationship` object. The order is important to maintain referential integrity: the order insures that the related objects are always persistent before the relationship is made persistent, thus avoiding a persistent state with references to non-existent objects.

If any of the contained objects being saved are themselves containers, the DMA client must also save each of their contained objects and relationship ob-

jects. The DMA client could implement this naturally via a recursive algorithm which rippled the saves down the containment subtree. This recursion would stop on any subtree element that is already persistent and has not been changed, and thus does not need to be made persistent again. This means that the DMA client must separately save each subtree of modified containers and contained objects.

Alternatively, a DMA client could batch all of these operations together in a single `IdmaBatch::ExecuteChanges()` call, taking care to order them as described above. `IdmaConnection::ExecuteChange()` or `IdmaBatch::ExecuteChanges()` on a `DirectContainmentRelationship` object will fail if either the containing or contained object is not yet bound to an underlying persistent object.

The process for creating referential containment relationships is identical to the above, except that an object of class `dmaClass_ReferentialContainmentRelationship` must be used instead of `dmaClass_DirectContainmentRelationship`.

3.6.5.2 Modifying a Container

To modify a container, a DMA client must first obtain a transient copy of the container object through navigation or search. From there, the DMA client can navigate to the containment relationship object(s) and the contained object(s). As described above, the client can then modify properties on any of those objects through the `IdmaEditProperties` interface. It can also change the ordering of the enumerations of contained and containing objects through the `IdmaRelationshipOrdering` interface on the relationship object(s), if the document space supports that interface. Saving the container and any new or changed contained objects and associated relationships is done as described earlier for new containers.

3.6.5.3 Deleting a Container

To delete a container's persistent form from the document space, a DMA client first must empty the container and remove it from any other container referencing it. The sequence of delete operations mirrors the sequence of operations used to create the containment relationship. At a minimum, this means deleting all containment relationship objects that reference the container via either Tail (for its contained objects) or Head (for objects containing the container to be deleted). (These can all be found by navigation.) The DMA client may also choose to delete the contained objects themselves, but this is not required by the DMA architecture.

To delete a persistent containment relationship object, a DMA client invokes the `IdmaConnection::SetDeletePending()` method on the corresponding transient DMA object. On the next call to `IdmaConnection::ExecuteChange()` (or `IdmaBatch::ExecuteChanges()`), the containment relationship object's persistent form is deleted, which effectively removes the contained object from the container. Once the container is empty and no longer contained directly or by

reference in any other container, it can be deleted as well by calls to `SetDeletePending()` and `ExecuteChange(s)`. A DMA client could choose to batch the emptying of a container with the deletion of the container in one call to `ExecuteChanges` after the appropriate sequence of `SetDeletePending()` calls.

A DMA client must similarly delete all containment relationship objects referencing a `Containable` object (such as a `DocVersion`) via `Head` before it can delete the `Containable` object itself.

3.6.5.4 Maintaining Referential Integrity

Document spaces are not required, but may choose, to enforce the referential integrity of their containers as a matter of policy. Document spaces may enforce a policy that persistent containment relationship objects are not allowed to reference non-existing objects. Document spaces that enforce referential integrity will return errors on methods that would result in a breach of referential integrity, such as deleting a child document without first deleting the direct containment relationship object referencing it. DMA clients must create, modify, and delete containers, their containment relationship objects, and contained objects in the order described above to avoid such errors.

If a document space has `DMAC_CONTAIN_DIR_REQUIRED` set to true in its `DocSpaceCapabilities` list, this means that it requires all `Containable` objects to be directly contained in parent containers at all times. No free-floating `Containable` objects are allowed. In this case, the ordering prescribed above for creating, modifying, or deleting a containment relationship will take the `Containable` object through an invalid persistent state in which it is not contained in any parent container. For instance, the order for creating a containment relationship calls for making the freestanding `Containable` object persistent *before* tying it into a container by making the containment relationship itself persistent.

Document spaces that elect to support this optional `DMAC_CONTAIN_DIR_REQUIRED` capability can work around the invalid state by supporting the `IdmaBatch::ExecuteChanges()` method and requiring its use for containment creation, deletion, and modification. This type of support allows the document space to insure against the indefinite persistence of an invalid state. DMA clients should be prepared to use `IdmaBatch::ExecuteChanges()` if they find it available and `DMAC_CONTAIN_DIR_REQUIRED` set to true; such a use avoids errors from the document space when the clients order method calls to preserve referential integrity at the expense of temporarily leaving a `containable` object uncontained.

The discussion above applies to the `DMAC_CONTAIN_REF_REQUIRED` capability as well.

3.6.6 Containment and Query

There are two styles of query in DMA -- (1) navigation, and (2) associative query. The navigational model of query is always handled the same way in DMA (i.e., by accessing properties of a DMA object the client has in hand using the `IdmaProperties` interface). The associative query model is also always handled the same way in DMA (i.e., by building a query object and invoking the `ExecuteSearch` method on a scope object). `ExecuteSearch` returns an interface to a result set object, and the result rows or “hits” of the query can be enumerated from it. The situation is the same for containment (i.e., since none are needed, there are no new ways of doing queries involving containment).

3.6.6.1 Navigational Queries

The starting point for navigating through the containment hierarchy of a document space is the enumeration of `InitialContainers` on the `DocSpace` object. Starting from the `DocSpace` object, use the `IdmaProperties` interface on that object to get to the `InitialContainers` enumeration object-valued property, and then use `IdmaEnumOfObject::GetNextObject()` to step through and select one of the containers in this enumeration.

Consider a direct containment example. Suppose you have an initial container object in hand, and you wish to enumerate the children of the container. Use the `IdmaProperties` interface on the container object to obtain the `Children` property. This property is an enumeration-of-objects property, and it can be used to enumerate the direct containment relationship objects between the container object in hand and the contained objects. The child direct containment relationship objects are enumerated by calling the `GetNextObject` method of the `IdmaEnumOfObject` interface of this property. For each direct containment relationship object so obtained, the `Head` property is obtained via its `IdmaProperties` interface. The value of this property is the contained object. (See `quexp3.cpp` in the query sample code in the appendix for an illustration of this simple loop.)

To enumerate the children of a container where the children are contained by reference (as opposed to direct containment), use the same method as discussed above except that the property `Containees` is substituted for the `Children` property.

Typically, the container object in the example above would be either in the class `dmaClass_Container` or one of its subclasses, although it could be an object of any class supporting the `Container` containment properties. The contained objects typically will be either of class `dmaClass_Containable` or one of its subclasses, but they could be objects of any class that happen to support the `Containable` containment properties.

To obtain the (unique) parent of an object contained by direct containment, one performs the following two steps. First, the `Parent` property of the contained object in hand is obtained via the `IdmaProperties` interface on it. This

gives you a direct containment relationship object. (If this is not the case, the object in hand is not contained via direct containment.) Second, the Tail property of the relationship object is obtained via the `IdmaProperties` interface on it; this is the parent object.

To obtain the container objects of an object contained by referential containment, one must write a loop similar to the loop for enumerating children. The biggest difference is that the `Children` property is replaced by the `Containers` property. Of course, the names of the variables should be changed to indicate that the parents are being enumerated in order to avoid confusion when reading the code. (See `quexp3.cpp` in the query sample code in the appendix for an illustration of this loop as well.)

3.6.6.2 Associative Queries

Associative queries in DMA involve building a query object with four main properties: (1) a select list, (2) a from list, (3) a query expression, and (4) an order by list. The query expression is a logical expression (i.e., an expression that returns a truth value), and it is perhaps best thought of as corresponding to a parse tree of a SQL-like query's "where" condition. The four main properties of a query object correspond to the four main clauses of a SQL select statement. (See `quexp2.cpp` in the query sample code in the appendix for an illustration of an associative query involving containers.)

Many document spaces use a Relational Database Management System (RDBMS) to store their document properties. When an RDBMS is used for this purpose, then for N:M relationships a natural way to model the data relationships in the database schema is to have a table where the rows of the relationship are stored, and to perform a join between tables as illustrated in the sample code in the appendix. (Referential containment is an N:M relationship.) This "relationship" table is also necessary when there is edge data for either direct or referential containment. However, not all document spaces support referential containment. Some support only direct containment with no edge data, and they do not rely on an RDBMS to do so; therefore, they do not have such a relationship table. In this case, the `DirectContainmentRelationship` object is synthetic. The `DirectContainmentRelationship` object being synthetic is merely an annoyance for navigational queries. However, it is more inconvenient when it appears in the parse tree. In its use in the parse tree, the document space has to scan the parse tree and figure out that it's not really there to join to and so has to massage the parse tree to bypass it.

DMA provides the `ParentContainer` property on objects of class `dmaClass_Containable` to solve this problem. The value of this property is the parent of the object in the direct containment relationship. It is not used for anything else. Its job is to bypass the relationship object between the child and its unique direct-containment parent. By joining the parent directly with the child, document spaces can bypass the intermediate relationship object.

The metadata for a Scope will indicate the supported methods for querying containment relationships. If `dmaClass_ContainmentRelationship` is marked as searchable, a DMA client may search for both direct and referential containment relationships in a single query. If not, separate queries are required. If the class `dmaClass_DirectContainmentRelationship` is marked as searchable, a DMA client may search for an object's parent by means of the intermediate `DirectContainmentRelationship` object. If the property `ParentContainer` is marked as searchable, a DMA client may search for an object's parent container directly without going through an intermediate `DirectContainmentRelationship` object.

3.7 DMA Content Model

3.7.1 Introduction

The DMA content model defines the properties, object classes, and interfaces that enable a DMA compliant application to access the content of a document, associate new content, modify content, and delete content from a document.

In this description, we refer to content as the information that can be used to produce one or more renderings of a document. For example, word processor and other office document files, image files, and HTML files are all considered content in DMA.

The DMA content model has been designed with the following characteristics:

- Support for document content is optional. A document space may choose not to support the properties, classes, and interfaces defined by the content model. Without content, a document is just a collection of properties. If a document space chooses to provide support for content, it has further levels of options as detailed in the content model.
- Complex documents consisting of single or multiple renditions can be easily represented.
- Document renditions made up of multiple components are accommodated.
- The content model is neutral to the format of the content.
- A client has the flexibility to store and retrieve document content via a stream or a general resource name in the form of an URL.
- Document spaces that can capture and manage content data as well as document spaces that maintain references to externally stored content data are accommodated by the content model.
- The content model allows creation, update and deletion of entire documents or individual components.

In addition to the primary document class (*DocVersion*), two classes of objects are used to represent document content in DMA, *Renditions* and *Content Elements*.

3.7.1.1 DocVersion

A DocVersion object is used to capture all the properties and sub-ordinate objects (renditions and content elements) to represent a single version of a document.

3.7.1.2 Rendition

A Rendition object is used to capture the properties and group the components required to produce one rendering of the document. In essence, renditions encapsulate the variations in content format. For example, one rendition could capture the information and file components for editing a document in Word, while another rendition could have a postscript file of the document for printing purposes.

3.7.1.3 Content Element

A Content Element object is used to capture the properties and provide access to a single content component. In essence, content elements encapsulate variations in storage and access to content data of a document. A content element would typically represent a file or a reference to a file for a particular component. Two sub-classes of Content Element, ContentTransfer and ContentReference are defined to represent content that is captured by a document space or maintained as a reference.

3.7.2 Modeling Document Content in DMA

The following example illustrates how a complex document consisting of a number of multiple component renditions can be represented in DMA.

Assume the following scenario:

- The editable document is made up of several Word (.doc) files being authored by a number of individuals.
- We wish to publish the document as a web of HTML files for access using a web browser.
- We also wish to provide a high resolution rendering of the document in PDF for printing and distribution purposes.
- The HTML and PDF renditions of the document are automatically generated by a tool that is invoked at the time of publishing by the editor of the document.

A possible scenario for a DMA compliant application to create the document described above would be as shown below:

- 1 Create a document object, either DocVersion or an appropriate sub-class of DocVersion.
- 2 Create a Rendition object for the Word format of the document.
- 3 Create a ContentTransfer object for each “.doc” file in the document. Assuming a document space that manages content, these files will be copied

to the document space. ContentReference objects can be created for any external references that need to be maintained.

- 4 Make the document and its content persistent in the document space.

The above document can now be made available for different individuals to update during the revision cycle.

The following could be done when it is time to publish the completed document.

- 1 Create a Rendition object for the HTML format of the document.
- 2 Generate the HTML files from the “.doc” files.
- 3 Create a ContentTransfer object for each file generated. ContentReference objects may need to be created to maintain any external references to content present in the Word rendition of the document.
- 4 Create a Rendition object for the PDF format of the document.
- 5 Create the PDF file from the “.doc” files (assume one PDF for the whole document).
- 6 Create a ContentTransfer object for the PDF file.
- 7 Add the PDF and HTML renditions to the document and make them persistent in the document space.

The diagram below shows the DMA objects that might make up such a complex document.

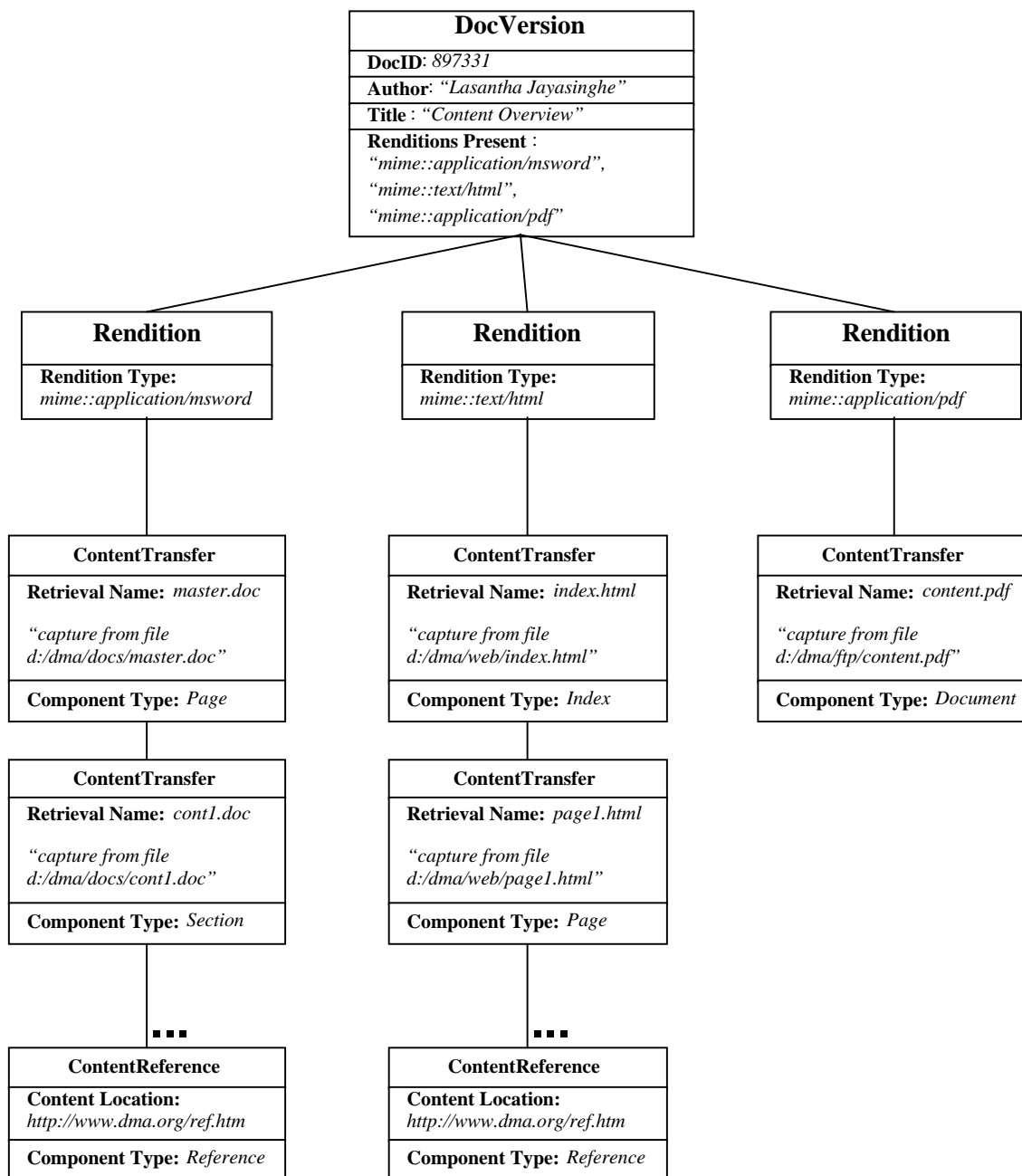


Figure 3-18: Sample DMA document

3.7.3 Content Creation

A DMA client application's first step in making document content persistent in a document space is to create a *DocVersion* object.

3.7.3.1 DocVersion Creation

As stated in the object model section, independently persistent objects (such as Doc Versions) must always be created from the document space in which the object will be persisted. Therefore a DMA client can invoke one of the following two methods to instantiate a new DocVersion object from the document space.

- 1 *IdmaObjectFactory::CreateObject ()* method on the document space, or
- 2 *IdmaClassDescription::CreateInstance ()* method on the class description object for a DocVersion, obtained from the document space in which the object is to be created.

Successful invocation of one of the above methods will result in a *scratchpad* DocVersion object being made available to the client application.

At this stage, the client will most likely set various property values on the new document object using methods on the *IdmaEditProperties* interface. The client then needs to create the necessary Rendition and Content Element objects in order to associate the content data for the document.

A subsequent call to either *ExecuteChange ()* or *ExecuteChanges ()* will make the scratchpad DocVersion object and any of its sub-objects persistent in the document space. (The transfer of content to the document space will occur at this time).

Note: In general, DocVersion objects are considered independently persistent. However, this is not mandatory and therefore DocVersion objects may be dependently persistent on another independently persistent object. It should be noted that Rendition and Content Element objects are always dependently persistent, typically on a DocVersion object. In this discussion, DocVersion objects are assumed independently persistent.

3.7.3.2 Rendition Creation

The DMA client application's next step in creating content for a document is to create the Rendition objects of the appropriate format to hold the content data. Renditions are objects of class *dmaClass_Rendition* or one of its sub-classes.

If a document space provides support for document content, then it must support the two content related properties, *dmaProp_RenditionsPresent* and *dmaProp_Renditions*, on *dmaClass_DocVersion* and all its sub-classes.

The `dmaProp_Renditions` property on a `DocVersion` is a list of Rendition objects. The client application can access this list the same as any other DMA “list of object” property. When a client application initially accesses this property on a new `DocVersion` it is supplied an empty list of objects. The client application would then construct as many Renditions that are required and insert them into this list using the `IdmaEditListOfObject::InsertObject` method.

Rendition objects can be created using one of the following methods.

- 1 `IdmaObjectFactory::CreateObject ()` method on the document space, or
- 2 `IdmaClassDescription::CreateInstance ()` method on the class description object for a Rendition, obtained from the document space in which the object is to be created.

Note: Since Rendition objects are dependently persistent objects, they could be created from another source other than the document space in which the document is being persisted. However, a document space is not required to accept such a Rendition object if it does not conform to the class description for Rendition objects in that particular document space.

Once a Rendition object is created, the DMA client application would set the `dmaProp_RenditionType` and other properties as appropriate on the Rendition. Typically, this would include creating the appropriate Content Element objects and inserting them into the Rendition object’s `dmaProp_ContentElements` list. After these operations are completed, the client application should have fully constructed Renditions with associated Content Elements; these Renditions can then be inserted into the `DocVersion`.

3.7.3.3 Content Element Creation

Before a DMA client application creates any content elements, it must determine what form of content capture is supported by the document space.

3.7.3.3.1 Determining the form of content capture

A document space supports content capture in at least one of two forms. Support for capture and management of content within the document space is provided via the class `dmaClass_ContentTransfer`, while support for maintaining references to content outside the control of the document space is provided via the class `dmaClass_ContentReference`.

A document space is allowed to support both forms of content capture. In this case, it is possible that a rendition may have a mixture of content data that is captured in the document space via a content transfer object, and some external references stored as content reference objects.

In order to determine what forms of content capture are supported by a document space, a client can either check the capabilities of the document space

(via its `dmaProp_DocSpaceCapabilities` value) or detect the presence of the above mentioned classes in the metadata for the document space. Based on this information and the form of the content data, the client application can decide which type of content element to create.

3.7.3.3.2 Creating ContentTransfer Objects

The client application would use either the `CreateObject` or `CreateInstance` method to create a new `dmaClass_ContentTransfer` object. It would then set `dmaProp_ComponentType` to some appropriate value in order to indicate what type of content element this is. Valid values for this property are dependent on the Rendition to which the content element belongs; DMA does not define these values. The client may also set a value for the `dmaProp_RetrievalName` property. Typically, this value is an URL format resource name where the content data currently exists. This property value could be used on subsequent access to the document content, to determine the original content resource name and/or the file name suffix possibly to launch an appropriate application.

After creating the content transfer object and setting its properties, the client then needs to determine the mode in which content will be supplied to the document space. Content capture and access can be done via Streams or Resource names in URL format. The `IdmaContentTransfer` interface provides all these methods.

To supply content as a stream the client application would create a stream on the content and invoke the method `SetCaptureStream ()` on the content transfer object. All document spaces must support this method of content capture. Alternatively, if supported, the client could invoke the method `SetCaptureResource ()` to identify content as a resource name in the form of an URL. Local file names – a common way of identifying document content, are made into URLs by prefixing the full path name with “file://”.

3.7.3.3.3 Creating ContentReference Objects

If content is supplied as a reference, the client application uses either the `CreateObject` or `CreateInstance` method to create a new `dmaClass_ContentReference` object. In addition to setting the `dmaProp_ComponentType` as discussed above, the client must set a value for the `dmaProp_ContentLocation` property. This value should be the resource name for the content data, in URL format. The document space may validate the URL, but does not need to provide any guarantees regarding the existence or accessibility of the content referenced by the resource name.

3.7.3.3.4 Making Content Persistent

Once the appropriate content elements are created, the client application would insert these objects into the Rendition object's `dmaProp_ContentElements` list property as discussed in the previous section. The document space

will capture and transfer the content data as appropriate when the parent DocVersion is made persistent.

Note: The client application must ensure that the resource name or stream supplied to a content transfer object exists until the DocVersion has been made persistent.

3.7.4 Content Access and Modification

The DMA content model provides two features that enable efficient content access and modification. First, Renditions and Content Element objects are bound to scratchpad objects only upon access to those specific objects due to the late binding rule. Second, actual document content is transferred to the client only upon invoking methods on the `IdmaContentTransfer` interface.

3.7.4.1 Locating and Accessing DocVersions

A DMA client can use either navigation (typically via containers), connection via OIID, or query to locate DocVersion objects in a document space.

The `dmaProp_RenditionsPresent` property is a system-generated list that reflects the rendition types that exist on a DocVersion at a given time. If a document space makes this property searchable, then it can be used to detect renditions that exist for a document via a DMA query. However, this property value and the `dmaProp_Renditions` list on the DocVersion are not kept synchronized in a scratchpad object. Therefore, it is not safe to assume that the Rendition object at a particular index in the `dmaProp_Renditions` list will have rendition type matching that of the corresponding index position in the `dmaProp_RenditionsPresent` list. The same rule applies to the `dmaProp_ContentElementsPresent` and `dmaProp_ContentElements` in the Rendition object.

Once a DocVersion object is located the client application would invoke the `IdmaDocSpace::ConnectObject ()` method to connect to the document and obtain a scratchpad copy of the persistent document.

3.7.4.2 Accessing Renditions and Content Elements

To access specific rendition and content element objects on a document a DMA client would use methods on the `IdmaListOfObject` interface on the `dmaProp_Renditions` and `dmaProp_ContentElements` list properties on the DocVersion and Rendition objects respectively.

3.7.4.3 Accessing Content Data

Once the client application has obtained a scratchpad copy of a specific content element, it can use an `IsOfClass ()` test to determine what type of content element (i.e. content transfer or content reference) was originally stored. If it is a content reference object, the client is responsible for accessing the content data directly by using the value of `dmaProp_ContentLocation`. Alternatively, if it

is a content transfer object, the client can then use one of a number ways to access content data as supported by the methods on the `IdmaContentTransfer` interface.

Delivering content data as a Stream must be supported by all document spaces that support content. The client would use the `GetStream ()` method to obtain a stream to the content stored in the document space.

Optionally, a document space may choose to support one or both of the following means of delivering content.

- The `CopyToResource ()` method to request the document space to copy content data to a resource name provided by the client
- The `GetResourceName ()` method to obtain a client accessible resource name for the content data

3.7.4.4 Modifying Content

In order to modify content, a client first needs to access the document and the relevant Rendition and Content Element objects as described above. Then the client application would use methods on the `IdmaEditListOfObject` interface to insert, replace, or delete objects in the Renditions or ContentElements list properties. Document space policy would dictate the type of modifications allowed on content objects and data.

Some document spaces may support direct replacement of existing content element data, by allowing `SetCaptureStream ()` or `SetCaptureResource ()` on already persisted objects.

3.7.4.5 Deleting Content

To delete document content or the document itself, the client would follow steps similar to those for modifying content.

If Renditions or Content Elements are deleted, the document space will delete the relevant objects from the persistent store for the particular document. If the `DocVersion` is deleted, then the document space will delete all persistent information about the document, including all its renditions and content elements. The DMA content model does not provide for sharing of content components among multiple objects, therefore if a document space has such a feature, it must not expose that fact to the DMA client application.

Note: The document space will not follow and delete the references supplied in ContentReference objects. In addition, a document space is not required to reclaim any storage at the time of deletion of content.

3.7.5 Content and the DMA Object Model

The following section discusses the classes, properties and interfaces defined for the content model. As stated previously, supporting content is optional for a document space. In addition to being indicated via the capabilities mechanism, the presence of the relevant classes, properties and interfaces indicates support for content by a document space.

The inheritance hierarchy of content classes is shown in the following diagram:

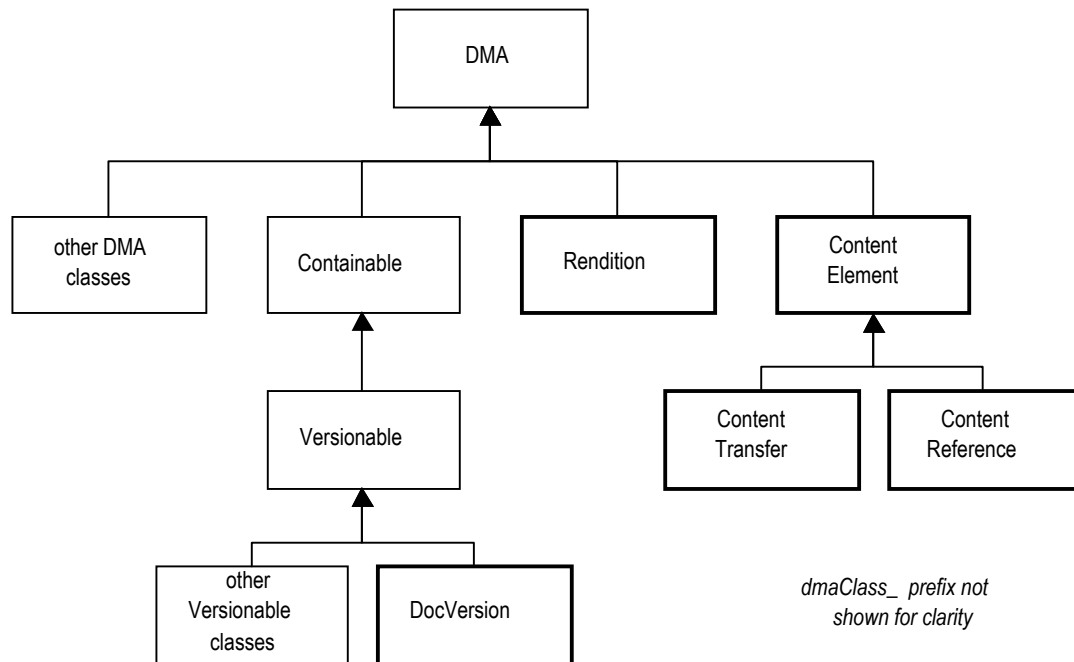


Figure 3-19: DMA Content Class Inheritance Diagram

3.7.5.1 DocVersion

The document version object is the base class for all document objects in DMA. This object contains properties about a document, as well as other sub-objects that represent content for the document. A document version object can participate in versioning operations as defined by the DMA versioning model.

A document version object can have one or more objects that represent different renderings (**Renditions**) of the document (for example, word-processor document files and the PostScript form of the document). The content of a document version is not associated directly, but rather through its renditions. Note: It is not mandatory for document version objects to have content.

The **DocVersion** class inherits from **dmaClass_Versionable**, and introduces the two properties, **dmaProp_RenditionsPresent** and **dmaProp_Renditions**, on the **DocVersion** class.

The `dmaProp_Renditions` property is a list of Rendition objects and can be accessed using DMA list access methods.

The `dmaProp_RenditionsPresent` property is a list of renditions present for the document. This list can be used to determine what renditions exist on a document either via a query or by inspecting the property on a specific document version.

3.7.5.2 Rendition

A Rendition manages all of the content elements associated with a particular rendering or representation of a document.

The class `dmaClass_Rendition` inherits from the base class `dmaClass_DMA` and introduces the `dmaProp_RenditionType` and `dmaProp_ContentElementsPresent` properties.

The `dmaProp_RenditionType` is a string property that differentiates the renditions of a document. These string values have the following syntax:

`<RenditionType_Space>::<Typename>`

The syntax and semantics of the `Typename` is specific to the rendition type namespace, `RenditionType_Space`. DMA 1.0 defines one `RenditionType_Space`, 'MIME', whose values are legal MIME types as defined by the IANA.

Examples values for `RenditionType` in the MIME namespace are: 'MIME::application/msword' or 'MIME::text/plain'.

Other rendition type namespaces may be defined in future versions of DMA.

The `dmaProp_ContentElements` property is a list of content elements on the rendition and can be accessed using DMA list of object methods.

The `dmaProp_ContentElementsPresent` property is a list of the component types of the content elements on the rendition and can be accessed using DMA list of string methods.

Due to the DMA content model being neutral to content format, the Rendition class has a minimal set of properties. However, it is anticipated that DMA implementations would sub-class Renditions to provide content format specific properties and behavior. For example, renditions that support the specific characteristics of Word, HTML and TIFF may be defined.

3.7.5.3 Content Element

This abstract base class represents an elementary content component. This class introduces the `dmaProp_ComponentType` property inherited by the other content element sub-classes.

The `dmaProp_ComponentType` is a string property that differentiates the content elements of a rendition. The syntax and semantics of this property is specified by the class of rendition associated with this content element; therefore, it is not defined in the DMA specification.

3.7.5.4 Content Transfer

This is a sub-class of content element that provides capture and access to document content managed within a document space.

The string property `dmaProp_RetrievalName` introduced by this class captures either the full name or suffix that the client would like the content to have upon retrieval for viewing or editing. This can be useful when retrieving content to determine the location to place the content and/or to determine the application to launch to handle the content (based on file extension).

Objects of class Content Transfer must support the *IdmaContentTransfer* interface.

3.7.5.4.1 IdmaContentTransfer

This interface provides methods to supply and access content data in DMA. The only mandatory form of content transfer is via a stream. This means that DMA clients and document spaces must provide support for supplying and retrieving content by implementing the *IdmaStream* interface.

3.7.5.4.2 IdmaStream Interface

This utility interface provides methods to perform read-only access to content data as a continuous stream of bytes. This interface inherits from *IUnknown* and introduces the two methods, `ReadStreamData ()` and `SetStreamPosition ()`. Some implementations might not support seeking into a stream, thereby making it a sequential stream.

This interface must be presented through a COM object, which need not be a DMA object. When content is inserted (via `SetCaptureStream ()`), the client application is responsible for instantiating an appropriate COM object. When content is retrieved (via `GetStream ()`), the document space implementation will instantiate the object.

3.7.5.5 Content Reference

This is a sub-class of content element, which references content outside the control of a document space.

The string property `dmaProp_ContentLocation` introduced by this class contains the reference to content data managed outside the document space. This value is a resource name in the format of an URL.

3.8 DMA Versioning Model

The DMA Versioning Model provides fundamental document-management operations. A document can be taken through a progression of successive versions, and the relationship among those versions can be tracked using DMA. Documents managed under the DMA versioning model can be updated and revised using standard check-in and checkout operations of the model.

The model in DMA 1.0 supports what is known as *linear versioning*. The different versions of a document are maintained as a series of individual document version objects. In this sequence, there is one instance that is designated as the current version. The model is organized in a way that allows extension to configuration management of document organizations (including branched versioning). However, the operations necessary to support other than linear series of versions are not specified for DMA 1.0.

The basic DMA 1.0 model for versioning presumes that there is some conceptual entity represented as a sequence of progressive versions. These versions may be defined in any number of ways, yet they each present a state of the conceptual entity at some point. There is at most one current version, which represents the current state of the conceptual entity. In this view, the versions constitute a possibly incomplete history of states of the conceptual entity.

In the DMA 1.0 specification, only Document Version objects are defined as well-known versionable objects, but the concepts can be generalized to allow versioning of additional well-known objects in the future. (The addition of application-specific versionable objects via sub-classing is available immediately.)

The DMA 1.0 specification does not require the Document Spaces to support versioning. The specification merely specifies how versioning is expressed for those Document Spaces which do support versioning. Additionally, the well-known objects used in DMA 1.0 solely to express versioning need not be supported by Document Spaces which do not support versioning.

The informal view of the DMA Version Model is illustrated in the diagram below. The different versions of a document can be thought of as states that once existed in the history of a conceptual entity. In the DMA 1.0 Version Model, the conceptual structure is represented by a configuration history that holds together the arrangement of the version states of the document. The Configuration History locates the Primary Version Series and the individual versionable objects are strung onto the series much like beads on a string. The connector between a place in a version series and a versionable object provides a Version Description for that particular occurrence of the versionable object.

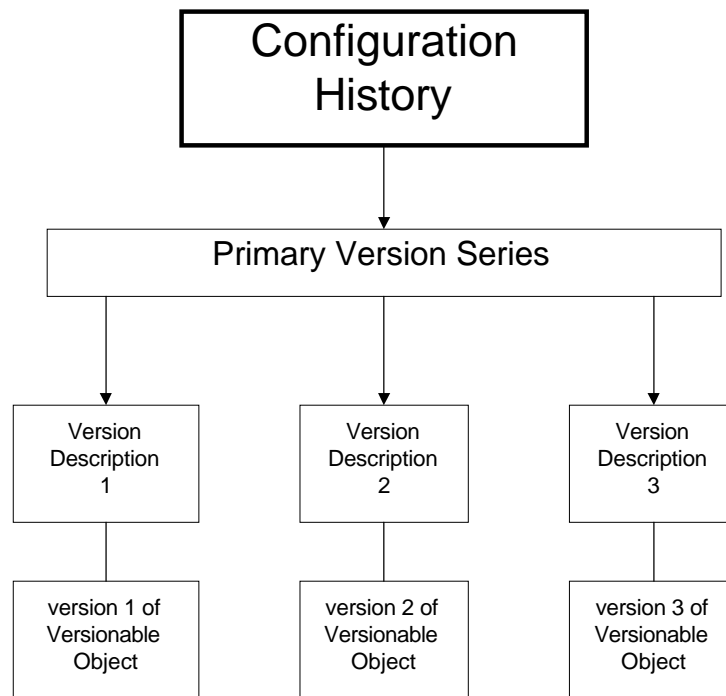


Figure 3-20: DMA Model of Versioned Objects (conceptual)

The DMA model allows the possibility that a versionable object might exist as a constituent of more than one Configuration History, or that it might occur in the same Configuration History in more than one way.

Because a versionable object may potentially occur in configurations in more than one way, the data that is specific to each occurrence is kept separate from the versionable object in a Version Description object. The Version Description object bridges a versionable object to a particular place in a configuration of versions and provides properties that are specific to that occurrence.

3.8.1 Objects Used for Versioning

If a document space supports the DMA versioning model, it will support (sub-classes of) the following objects:

- DMA Configuration History
- DMA Version Series
- DMA Version Description
- DMA Reservation

The objects listed above are used, in addition to the individual versionable objects themselves, to manage the organization of conceptual-entity version se-

ries and the check-in and the check-out process for versions of conceptual entities.

If the DMA Versioning Model is not supported by a particular document space as a matter of policy or implementation, then objects of these kinds will not be accepted in that space, and class descriptions for the objects will not appear.

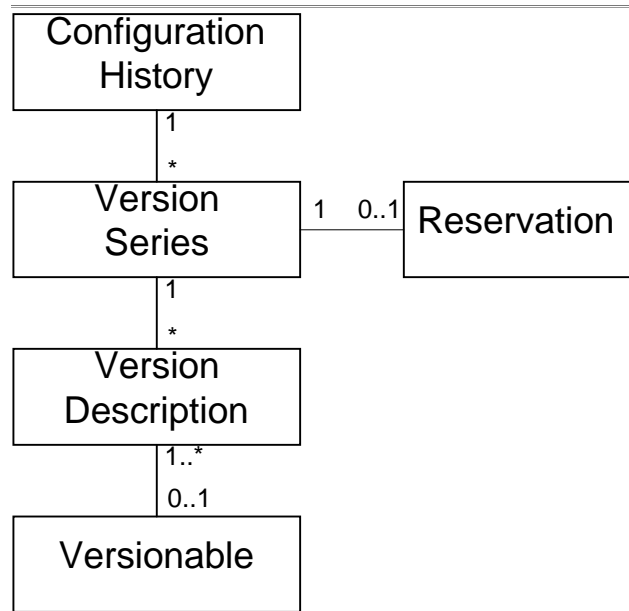


Figure 3-21: DMA Version Management Navigation Model

In the DMA 1.0 version management navigation model, there are four levels to a (conceptual) versioned object:

- The Configuration History is the top-level object that “holds” the configuration of versions. It provides a unique entry into the overall configuration. (The entire configuration can be navigated via entry at the Configuration History object.)
- The Version Series is a DMA object that carries the configuration of a single “line” of versions that are viewed as generally having a progressive history. There is a Primary Version Series representing the top-level version sequence of the configuration. In linear versioning, there is exactly one Version Series. In more-complex configurations, additional Version Series are used to provide branches and subordinate configuration progressions beneath the Primary Version Series.
- The Version Description is the connection from a place in a Version Series to a particular versionable object. The Version Description accounts for the occurrence of that object version in that place in a particular version series of a configuration.
- The Versionable Object is some instance of an object (e.g., a Document Version object) that represents the conceptual entity at a particular point.

The objects that support versioning are shown in the following table:

Table 3-4 Versioning Objects

Feature	Object(s)
Linear Versioning	Version Series, Version Description, and Reservation
Branch Versioning	Configuration History, Version Series, Version Description, and Reservation
Threaded Versioning	Version Description, and Reservation

The following diagram shows the inheritance of the classes used for versioning in this proposal. (The versioning classes are outlined in bold lines.)

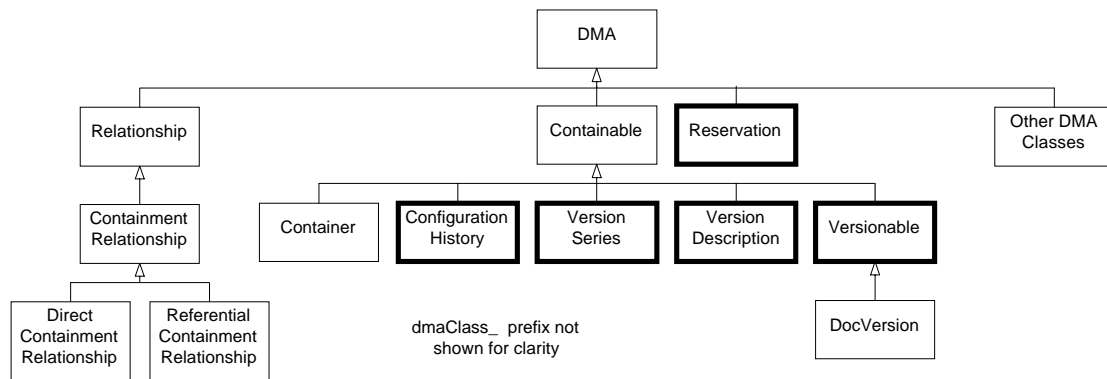


Figure 3-22: DMA Version Management Inheritance Diagram

3.8.2 Types of Versioning

Three types of versioning are mentioned in this proposal: Linear Versioning, Branched Versioning, and Threaded Versioning. The versioning types are described and discussed in the following sections.

3.8.2.1 Linear Versioning

Linear versioning includes configurations in which successive versions are organized in a sequence or series. There is one member of the series, usually the latest one, which is recognized as the Current Version of the series.

Version Series objects implement linear series of versioning. Each Version Series can be viewed as the recorder for the series and its state.

Version Series do not mention the versionable objects directly; instead, a Version Series records a sequence of Version Description objects. Each Version

Description links to a single versionable object. The Version Description provides a unique connection between a Version Series and a versionable object.

The Version Descriptions attached to a version series provide any descriptive application information about how the versionable object comes to be incorporated in the version series in that place. The Version Description is valuable for carrying descriptive information about how the object occurs in the particular Version Series. For example, the Version Description subclass used will usually provide a label (e.g., “5” or “2” or “C” or “draft 5” or “final”) by which the object is known relative to the containing series. In addition, there may be information about the authority under which the Version Description was added and when it was done.

The Version Description is valuable as a separate object, whether or not a versionable object can be attached to more than one Version Description (see “Threaded Versioning,” below). The Version Description can be subclassed independent of versionable objects to add more version-related application information without forcing more subclassing on the versionable objects themselves.

In addition, DMA 1.0 defines classes of objects, such as the DMA Document Version, that are designated as *versionable*. Versionable objects carry certain version-management properties that are required for the object to be managed as a state of a versioned object. Versionable objects also offer the *IdmaVersionable* interface. In a Document Space that does not support versioning, these properties take on specific default values and the *IdmaVersionable* interface need not be offered.

In the DMA 1.0 Versioning model, versionable objects are not required to be versioned. In the model, it is permissible for versionable objects to exist as standalone objects not reachable by any Version Description. (Applications

and document space policy mechanisms can restrict the model as appropriate.)

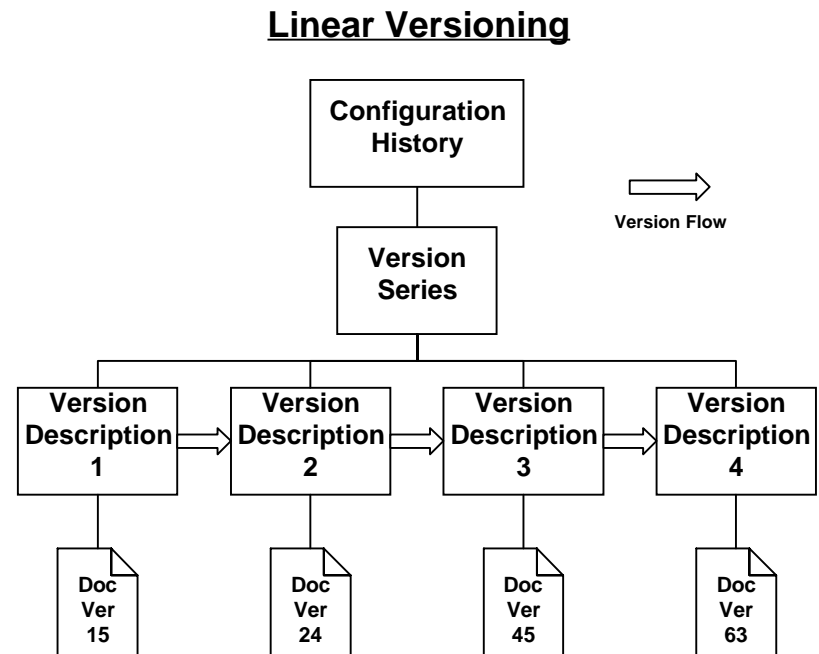


Figure 3-23: Linear Versioning

3.8.2.2 Branch Versioning and Complex Configurations

Branch Versioning is an extension beyond the basic linear versioning of DMA 1.0 in which a conceptual entity's history of configurations requires multiple Version Series intertwined in a variety of ways. Operations for creating branches of versions and more-complex configurations are not provided in the DMA 1.0 Versioning specification, but the impact on navigation is defined.

The DMA 1.0 Versioning model anticipates complex configurations by assuming that a Configuration History object might list the presence of more than one Version Series.

In addition, the DMA Version Description provides for one or more branches being taken from that point in a series. The DMA Version Description also provides for one or more Version Series merging into the current series at the Version Description point. These are the only ways that complex configurations are defined to be apparent to DMA 1.0 applications.

To permit scaling from linear versioning to more complex versioning and configuration-management models beyond DMA 1.0, Configuration History is always used, even for linear versioning configurations having a single Version

Series. In this arrangement, linear versioning occurs as simply a restricted case of a more complex configuration.

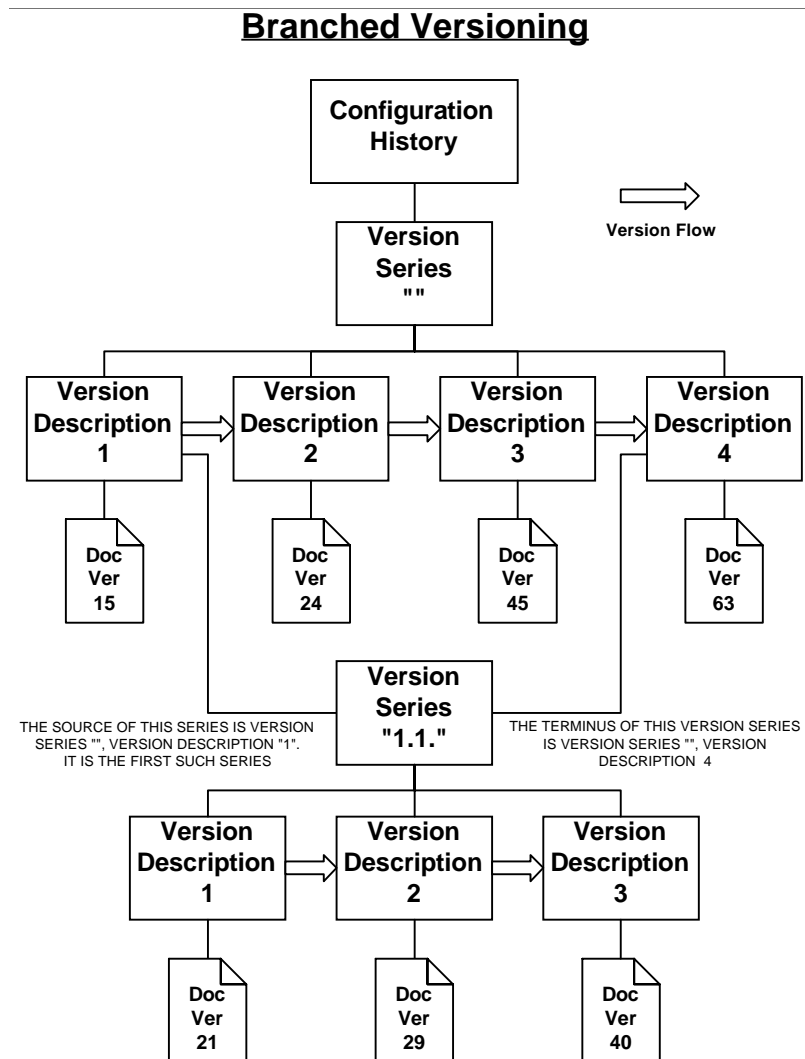


Figure 3-24: Branched Versioning

3.8.2.3 Threaded Versioning

The DMA 1.0 Versioning model explicitly allows for a versionable object to exist as a version of one or more conceptual entities and in one or more ways. There is no model-inherent limitation on how many version descriptions may lead to the same versionable object. Stated another way, the versionable object may be threaded through more than once by the same or different Version Series.

DMA retains this provision to allow configurations of objects for different purposes. For example, there might be a special version series of only official releases of a publication, and other version series might include intermediate internal revisions and proposals. Similarly, some version descriptions might relate to stages of a workflow that a versionable object is taken through, yet

the various reviews, approvals, and transmittals might all refer to the same versionable object.

Whether threaded versioning is permitted in general or on a case-by-case basis will depend on policies honored by document spaces and availability of applications that operate successfully within those policy constraints.

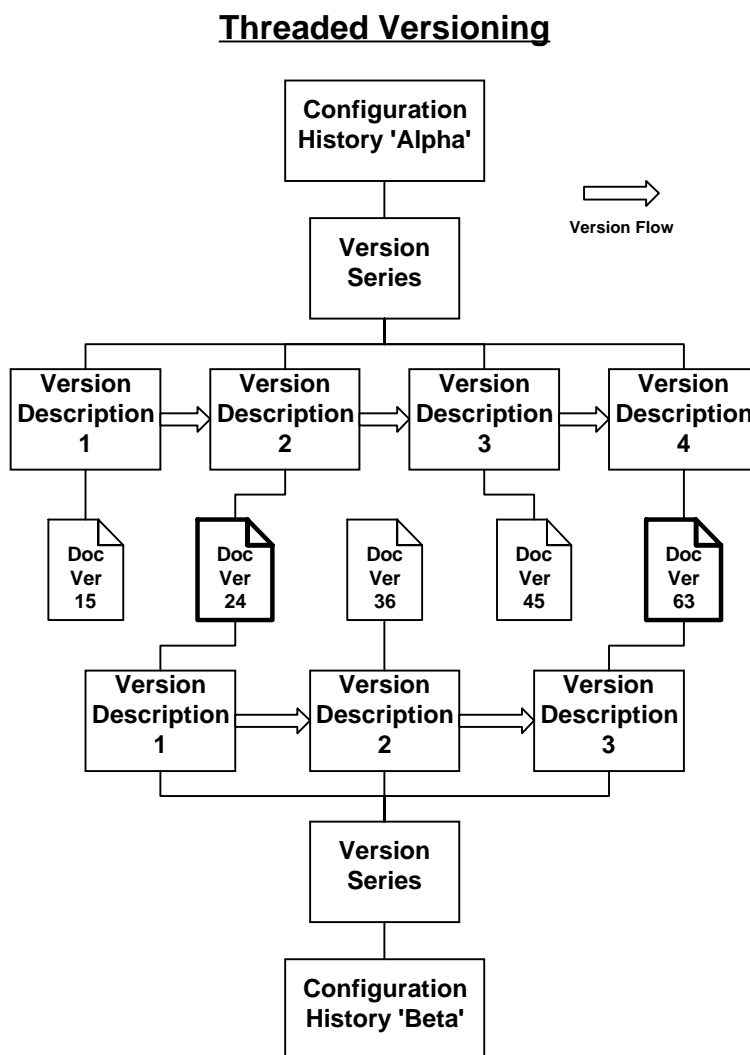


Figure 3-25: Threaded Versioning

3.8.3 Basic Characteristics of DMA Versioning

The DMA Versioning model is based on the idea that the configurations of material are being managed without regard to variation among the different versions. The DMA 1.0 Versioning model specification does not specify any intrinsic relationship between versionable objects that are attached to version descriptions.

The basic check-in model simply adds versionable objects to a version series. There is no indication about how that versionable object is related to any others in the series nor what the implications might be of the versionable object also being included (via version description) into multiple version series.

Similarly, the available “check-out” operation is the reservation of the right to check in the next new member of a series. There is no requirement that the new member have any particular relationship to others of the series.

Any relationship between the derivation of an object and its occurrence in a particular version series is a function of applications and possibly some constraints implemented by policy mechanisms of document spaces.

Applications may operate quite differently. Some application may impose strict relationships between versionable objects and how new ones are derived for check-in into version series. Such applications can honor those policies by using the DMA 1.0 versioning operations as building blocks of the more-specific form of document versioning that is delivered to users of a DMA system.

3.8.4 DMA Versioning Operations

In DMA 1.0, the following versioning operations are supported:

- **SetReserveNext** and **CheckOutNext**: Both of these operations reserve the (exclusive) right to add a new version at the end of a Version Series. In addition, CheckOutNext makes a copy (clone) of the Current Version of the Version Series for possible modification (done mainly to allow for optimizations in an implementation).
- **SetCheckIn**: This operation creates the next Version in a Version Series and makes it the Current Version. It also removes the reservation created by SetReserveNext or SetCheckOutNext.
- **SetRevoke**: This operation cancels a previous SetReserveNext or SetCheckOutNext.

Note: These versioning operation names listed above are not the exact method names, and the operations described are performed in two steps as dictated by the persistence model of DMA.

3.9 DMA Internationalization and Localization

The DMA architecture supports the internationalization and localization of document management applications and systems through (1) the use of the Unicode universal character set encoding, (2) a locale inspired mechanism for specifying a client's language, (3) support for multiple text ordering methodologies, and (4) a locale independent date/time representation.

3.9.1 Internationalization affected DMA Data Types

The only data type utilized by the DMA COM API for representing possibly unconstrained textual data is the `DmaString` data type (actually only `pDmaString` and `ppDmaString` references to this data type are utilized).

The DMA `DateTime` property type is defined as a syntactically constrained `DmaString`. This section defines the syntax that DMA will employ through its API for `DateTime` properties in a manner that is unambiguous with respect to time zones, language, and cultural conventions.

DMA Object Instance Identifiers (OIID) are represented using a syntactically constrained `DmaString` data type. DMA OIIDs are not viewed as localizable textual data by the DMA Internationalization and Localization model, and as such are not discussed further in this section.

3.9.1.1 `DmaString` Data Type

DMA follows the Microsoft OLE Automation conventions with regards to its string data type and utilizes a Microsoft `BSTR` conformant in-memory data representation for `DmaStrings`.

A `BSTR` can be thought of as a pointer to a null terminated array of `wchar_t` characters. The `BSTR` character data array is preceded in memory by a 32-bit integer that contains the length of the `BSTR` character data array. This feature allows efficient marshalling of the `BSTR` since the character data array length is explicit. In many cases, the `BSTR` can be treated as a normal null terminated string since the pointer is to the first element in the character data array. DMA adopted the `BSTR` data representation to facilitate the introduction of DMA language bindings for languages such as Visual Basic and Java.

DMA specifies that four macros be defined in the DMA header files for manipulating DMA's `BSTR` conformant strings. The macro names are listed below:

- `DMA_CREATE_STRING`
- `DMA_FREE_STRING`
- `DMA_GET_STRING_TEXT`
- `DMA_GET_STRING_CHAR_COUNT`

These macros are defined in the Macros Reference section of this specification.

3.9.1.2 DateTime Properties

DMA utilizes a locale-independent date/time format. DMA clients are expected to transform DMA date/time values into locale and platform specific representations as required for user presentation. A `DmaDateTime` is a specialization of a `DmaString` with the following syntax derived from ISO 8601:

YYYYMMDDThhmmss[,f]Z

where:

YYYY	represents a four-digit year number according to the Gregorian calendar
MM	represents a two-digit month in the range 01-12
DD	represents a two-digit day in the range 01-31
hh	represents a two-digit hour in the range 00-23
mm	represents a two-digit minute in the range 00-59
ss	represents a two-digit second in the range 00-59
f	represents decimal fractions of a second to arbitrary precision

The literal "T" separates the date and time components. The optional literal "," (comma) separates the time and fractional seconds components. The literal "Z" is mandatory and indicates that the time is represented in Coordinated Universal Time (also known as UTC or GMT).

Midnight is represented by a time (hhmmss) of 000000 and indicates the start of the specified date. Applications desiring to store only dates using `DateTime` properties should specify a time component of midnight.

A Document Space implementation may not be able to store `DateTime` values persistently with the precision specified by the caller. DMA does not require that a Document Space implement `DateTime` persistence with any specific precision. However, the Document Space must accept and return `DateTime` values per the syntax described in this section.

3.9.2 Character Set Encodings

DMA 1.0 specifies that the Unicode (UCS-2) character set encoding must be supported by all DMA 1.0 compliant implementations. Support for other character set encodings is neither encouraged nor discouraged by the specification. (The DMA architecture is based upon principles that enable supporting additional character sets encodings.)

DMA 1.0 requires that all `DmaStrings` passed through COM interfaces on a System Manager object, or objects derived from it, must utilize a common character set encoding. The common character set encoding is an attribute of a DMA System and is bound to a System Manager object instance at creation time. DMA API callers (both applications and service objects) are responsible for presenting and accepting `DmaString` data in the System Manager's common character set encoding.

In order to support use in countries other than the United States, DMA supports the expression of characters utilized by the local languages. Latin based languages draw from a relatively small character set and are generally encoded one character per byte whereas Asian languages draw from much larger character sets and require multiple byte encodings. In order to support non-ASCII character set encodings, the DMA architecture has the following characteristics:

1 Character set neutrality with regards to the handling of `DmaStrings`:

Character set neutrality means that DMA will not require the use of any specific code point to represent a character nor shall it construe that a code point can be represented with a single byte of storage.

The DMA System Manager and middleware pass `DmaString` data between a client and a System or DocSpace implementation, respectively, in whole, and as such should be character set neutral. However, queries on merged scopes may require the merging of result sets from individual document spaces possibly requiring that the DMA middleware compare `DmaString` data to determine relative order. DMA's Service Object mechanism is utilized to support the installation of third-party provided `DmaString` text ordering implementations.

2 Unambiguity with regards to character set encodings:

A client application must be able to determine with which character set encoding a `DmaString` is encoded. This capability does not require that each `DmaString` be tagged with a character set encoding ID, only that within the context of use the character set encoding is unambiguous.

3.9.2.1 DMA Character Set Encoding Identifiers

A DMA character set encoding identifier is represented using a `DmaInteger`. DMA will not administer a registry of character set identifiers, instead DMA character set encoding identifier values will be drawn from the Internet Assigned Number Authority (IANA) Character Set Registry (which is utilized by various Internet standards including MIME and HTTP). The IANA Character Set Registry defines a name and possibly several aliases for each character set. The IANA Character Set Registry also defines unique integer values, (referred to as the MIBenum value) for these character sets. DMA character set encoding identifiers utilize the IANA Character Set registry MIBenum values.

DMA character set encoding identifier values for some "commonly occurring" character sets are enumerated in the following table.

Table 3-5 DMA Character Set Encoding Identifier values

Character Set Encoding Standard (Description)	DMA Character Set Encoding Identifier Value (IANA MIBenum Value)
ISO-10646-UCS-2 (Unicode)	1000
ANSI X3.4-1968 (US ASCII)	3
ISO 8859-1:1987 (Latin1)	4
ISO 8859-2:1987 (Latin2)	5
ISO 8859-3:1987 (Latin3)	6
ISO 8859-4:1987 (Latin4)	7
ISO 8859-5:1988 (Latin/Cyrillic)	8
ISO 8859-6:1987 (Latin/Arabic)	9
ISO 8859-7:1987 (Latin/Greek)	10
ISO 8859-8:1987 (Latin/Hebrew)	11
ISO 8859-9:1989 (Latin5)	12
Shift JIS (MS Kanji)	17
EUC Packed Format for Japanese (EUC-J)	18
EUC-KR (KS C 5861-1992, RFC 1557)	38
KS C 5601-1987 (Korean, RFC 1345)	36
ISO-10646-UCS-4	1001

DMA 1.0's standard character set encoding, Unicode (ISO-10646-UCS-2), has a DMA character set encoding identifier value of 1000.

3.9.3 Language Support

In order to support implementation of global enterprise document management systems that serve clients whom use different languages, potentially located in different countries, the DMA architecture allows implementations to be capable of presenting textual information to a client in a language specified by the client. Furthermore, a client can determine which languages are supported for the various components of a DMA Document System. (These requirements are referred to collectively as the localization requirement.)

DMA defines a language via a locale name mechanism. Language is the primary attribute that DMA 1.0 will infer from a locale name. A locale name will consist of a *<language, subtab>* two-tuple and will be represented as a DmaS-string value syntactically conformant with an Internet language tag value as de-

defined per RFC 1766. Locale names will normally consist of a two-letter ISO 639 language abbreviation, a hyphen, and a two-letter ISO 3166 country code (e.g. en-US, English in the United States), although RFC 1766 allows other minor variations. Per RFC 1766, language and subtag components consist of alphabetic characters drawn from the US-ASCII character repertoire (a-z, A-Z). A DMA locale name value will use the character set encoding of the DMA system in which context it is utilized.

DMA addresses the localization requirement by (a) requiring that a DMA system implementation and DMA document spaces provide the client with one or more supported locale names and (b) allowing for the client to specify one of the supported locale names when instantiating a System or DocSpace object.

DMA imposes no requirements on a system or document space implementation with regards to the degree to which an implementation supports a locale. A client's specification of a locale when instantiating an IdmaSystem or IdmaDocSpace object simply expresses the client's preference for resource and capability localization.

The DMA System Manager implementation must be capable of providing the client with localized System names and descriptions.

The DMA System implementation may utilize the locale to provide the client with localized "Descriptive Text" and "Display Name" property values for the ClassDescription and PropertyDescription objects associated with the system object. The DMA System implementation may also utilize the locale to present localized message text (returned by the IdmaSystem::GetResultCodeDescription method).

The Document Space Service Object implementation may utilize the locale to provide the client with localized "Descriptive Text" and "Display Name" property values for the ClassDescription and PropertyDescription objects associated with Document Space objects.

The Document Space Service Object may also utilize the locale to otherwise modify the capabilities of a Document Space, such as providing the client with Ordering IDs meaningful to that locale only.

3.9.4 System Manager Internationalization

The System Manager provides clients with the capability to bootstrap into a DMA object model environment by connecting to a DMA System. The System Manager also provides the capability to register system and text ordering service objects.

A DMA System may be registered more than once with the DMA System Manager with distinct character set encodings. This mechanism enables a logical system to be registered, (for example, utilizing the Unicode and Shift-JIS character set encodings). This use of multiple System registration indicates that

such Systems are to be treated identically other than with respect to character set encoding. For example, they can be thought of as providing access to the same collections of documents.

3.9.5 Text Collation

DMA text collation (ordering) occurs only within the context of a DMA query and is described in detail in the Query section of this document. DMA middle-ware only deals with text collation when performing queries on merged scopes involving textual properties. Text Ordering Service Objects may be registered with the System Manager for use by System Service Object implementations. Text Ordering Service Objects implement one or more text collations (orderings) specific DmaString comparisons and are intended to be utilized by DMA System Service Object implementations to perform full- or merge-sorts on query result rows.

3.9.5.1 Well-known DMA Text Collation Sequences

DMA 1.0 defines the following well-known text collation sequences:

Name	Define Name	Description
Case Sensitive Code Point Comparison	<code>dmaCollation_CaseSensitiveCodePoint</code>	Strict code-point based comparison
Case Insensitive Code Point Comparison	<code>dmaCollation_CaseInsensitiveCodePoint</code>	Compares strings case-insensitively and locale independently.
Case Sensitive Lexicographical Comparison	<code>dmaCollation_CaseSensitiveLexicographic</code>	Compares using lexicographic ordering in the current locale.
Case Insensitive Lexicographical Comparison	<code>dmaCollation_CaseInsensitiveLexicographic</code>	Compares based on lower case lexicographic ordering in the current locale.

3.9.5.1.1 Detailed Descriptions

This section provides a detailed discussion of the text collation sequences listed above.

- Case Sensitive Code Point Comparison {`dmaCollation_CaseSensitiveCodePoint`}

Compares and orders strings strictly according to the code point values of the characters. This is a locale-independent collation sequence. Under the Unicode character set encoding, comparison of strings using this collation sequence delivers results exactly as for the ANSI C runtime function `wcscmp`.

- Case Insensitive Code Point Comparison {dmaCollation_CaseInsensitive-CodePoint}

Compares and orders strings according to the code point values of the characters after (logically) converting them to lower case. Ordering is based on the numerical values of the lower case character codes rather than the lexicographical order of those characters. Therefore, this is a locale-independent collation sequence. Under the Unicode character set encoding, comparison of strings using this collation sequence delivers results exactly as for the Microsoft C runtime function `_wcsicmp`.

- Case Sensitive Lexicographical Comparison {dmaCollation_CaseSensitiveLexicographic}

Compares and orders strings according to the lexicographical ordering of the characters as determined by the current character set encoding and locale. Under the Unicode character set encoding, comparison of strings using this collation sequence delivers results as for the ANSI C runtime function `wscoll`.

- Case Insensitive Lexicographical Comparison {dmaCollation_CaseInsensitiveLexicographic}

Compares and orders strings by (logically) convert to lower case and then applying the lexicographical ordering as determined by the current character set encoding and locale. Under the Unicode character set encoding, comparison of strings using this collation sequence delivers results exactly as for the Microsoft C runtime function `_wcsicoll`.

4 DMA Reference

These sections provide the definitions for the classes that are defined for the DMA object model, the C functions used to bootstrap a DMA System, the COM interfaces supported by DMA, macro functions supplied by DMA, the list of possible return codes from DMA interfaces, the query join operators, the other query operators, the base DMA data types, and conformance.

- Object Reference Table of Contents
- Interfaces and Methods Table of Contents
- Macros
- Return Codes
- Join Operators
- Query Operators
- Data Types
- Conformance

4.1 Objects and Properties

4.1.1 DMA Object Reference Column Descriptions

The class summary table describes the characteristics of DMA classes. Each column describes a particular characteristic. Column definitions are as given below. The class summary table has one row for each class defined by the DMA specification:

- **Class Id:** The value of this column is the name of the DMA class Id variable. The class Id variables are declared in DMA header files, and they are statically initialized variables of type Dmald. NOTE: The name of the class Id variable is formed from the class name by discarding embedded blanks and prefixing the result with "dmaClass".
- **Superclass:** The classes of DMA (including both the classes defined in the DMA specification and the classes defined by a particular Document Space) are organized into a single inheritance hierarchy. The value of this column is the name of the immediate superclass of the class being described. The root class of the hierarchy is DMA, and it is the only class that does not have a (unique) superclass.

The detailed description of each DMA class begins by repeating the class attributes described in the preceding paragraph. The detailed description of each DMA class also includes a property summary table. The property summary table has a row per property that summarizes a certain set of characteristics of the properties of the class. The following are the meanings of the column headings in the property summary table:

- **Name:** The value of this column is the name of the property. NOTE: A detailed description of each property follows the property summary table. The descriptive text for each property begins with a bold face heading composed of the property name followed by the name of the property Id variable. As with the class Id variable's, the property Id variables are declared in DMA header files, and they are statically initialized variables of type Dmald. The property Id variable name is formed from the property name by discarding embedded blanks and prefixing the result with "dmaProp_".
- **Impl. Required:** If the value of this column is "Yes", then all implementations of the DMA API must implement this property, and there must be an element in the Property Description List of the Class Description object for the property. Otherwise, the property is part of an optional feature (e.g., versioning or containment). If the optional feature involving the property is *not* implemented, then there *must be no element* for the property in the Property Description List of the Class Description object for the class (if the Class Description exists). Conversely, if the optional (or required) fea-

ture *is* implemented, then there *must* be an element in the Property Description List of the Class Description object for the class.

- **System Gen'ed:** If the value of this column is "Yes", then the value of this property is generated by the implementation of the DMA API as a side effect of some DMA method call.
- **Read-Only:** If the value of this column is "Yes", then the client can not set the value of the property directly using the `IdmaEditProperties` interface.
- **Value Required:** If the value of this column is "Yes", then the property must have a legal value, *assuming the property is supported* by the implementation. If the property is a list or an enumeration, "having a legal value" means that the list or enumeration (which must be present as the value of the property in all circumstances) must contain at least one element.
NOTE: When an attempt is made to obtain the value of a property (via a method in the `IdmaProperties` interface) that is currently not set to a legal value, a `DMARC_VALUE_NOT_SET` error is returned. A property can have its value removed (i.e., nulled out) by calling `DeletePropValByID` or `DeletePropValByIndex` in the `IdmaEditProperties` interface. It is possible that "Impl. Required" is *not* "Yes", but "Value Required" is "Yes". This can happen if the property is part of an optional feature, because "Yes" for "Value Required" assumes that the feature is supported by the implementation involved.
- **Type:** The value of this column is the base datatype of the property. The possible base datatypes are Integer32, Float64, String, Binary, Boolean, DateTime, ID, and Object. There are macros in the DMA header files for each of the base data types.
- **Cardinality:** The value of this column indicates whether the value of the property is a "Scalar", "List", or "Enumeration". NOTE: A single instance of a value of type String, Binary (i.e., array of bytes), DateTime, or Id is considered to be a Scalar.
- **Required Class:** If the base datatype is "Object", then the value of this column designates a class such that the object must be of that class or one of its subclasses. NOTE: If the class of the object is not constrained, then the value of this column is "DMA", which indicates the DMA base class, DMA. If the base datatype is not "Object", then there is no value for this column, since any value would be meaningless.

4.1.2 COM Objects Reference

Object Name	Impl. Required
SystemManager	-

4.1.2.2 SystemManager

The System Manager is a COM object that is primarily used to enumerate and connect to DMA System objects.

Class ID: SystemManager

Superclass: none

4.1.2.2.1 Interfaces

- IUnknown
- IdmaObjectFactory [Optional]
- IdmaOIID
- IdmaServiceRegistry [Optional]
- IdmaSystemManager

4.1.2.2.2 Properties

None.

4.1.2.2.3 Description

The System Manager object is not a DMA object, in that it does not support the IdmaObject or IdmaProperty interfaces. It does not, therefore, participate in the DMA Class hierarchy nor is it self-describing. Client applications can use the dmaConnectSystemManager API function to get an interface on a System Manager object.

4.1.2.2.4 Property Descriptions

No Properties Defined for This Object.

4.1.3 DMA Objects Reference

Class ID	Superclass
dmaClass_ClassDescription	Metadata
dmaClass_ConfigurationHistory	Containable
dmaClass_Containable	DMA
dmaClass_Container	Containable
dmaClass_ContainmentRelationship	Relationship
dmaClass_ContentElement	DMA
dmaClass_ContentReference	Content Element
dmaClass_ContentTransfer	Content Element
dmaClass_DirectContainmentRelationship	Containment Relationship
dmaClass_DMA	
dmaClass_DocSpace	DMA
dmaClass_DocVersion	Versionable
dmaClass_Enumeration	DMA
dmaClass_EnumerationOfObject	Enumeration
dmaClass_List	DMA
dmaClass_ListOfBinary	List
dmaClass_ListOfBoolean	List
dmaClass_ListOfDateTime	List
dmaClass_ListOfFloat64	List
dmaClass_ListOfId	List
dmaClass_ListOfInteger32	List
dmaClass_ListOfObject	List
dmaClass_ListOfString	List
dmaClass_Metadata	DMA
dmaClass_PropertyDescription	Metadata
dmaClass_PropertyDescriptionBinary	Property Description
dmaClass_PropertyDescriptionBoolean	Property Description
dmaClass_PropertyDescriptionDateTime	Property Description
dmaClass_PropertyDescriptionFloat64	Property Description
dmaClass_PropertyDescriptionId	Property Description
dmaClass_PropertyDescriptionInteger32	Property Description
dmaClass_PropertyDescriptionObject	Property Description
dmaClass_PropertyDescriptionString	Property Description

Class ID	Superclass
dmaClass_Query	DMA
dmaClass_QueryConstant	Query Node
dmaClass_QueryConstantBinaries	Query Constant
dmaClass_QueryConstantBinary	Query Constant
dmaClass_QueryConstantBoolean	Query Constant
dmaClass_QueryConstantBooleans	Query Constant
dmaClass_QueryConstantDateTime	Query Constant
dmaClass_QueryConstantDateTimes	Query Constant
dmaClass_QueryConstantFloat64	Query Constant
dmaClass_QueryConstantFloat64s	Query Constant
dmaClass_QueryConstantId	Query Constant
dmaClass_QueryConstantIds	Query Constant
dmaClass_QueryConstantInteger32	Query Constant
dmaClass_QueryConstantInteger32s	Query Constant
dmaClass_QueryConstantString	Query Constant
dmaClass_QueryConstantStrings	Query Constant
dmaClass_QueryJoinOperator	Query Nonterminal Node
dmaClass_QueryNode	DMA
dmaClass_QueryNonterminalNode	Query Node
dmaClass_QueryOperandDescription	DMA
dmaClass_QueryOperator	Query Nonterminal Node
dmaClass_QueryOperatorDescription	Metadata
dmaClass_QueryOrderByNode	Query Node
dmaClass_QueryProperty	Query Node
dmaClass_QueryResultRow	DMA
dmaClass_QueryResultSet	Enumeration
dmaClass_QueryRoot	Query Node
dmaClass_QuerySearchableClass	Query Node
dmaClass_ReferentialContainmentRelationship	Containment Relationship
dmaClass_Relationship	DMA
dmaClass_Rendition	DMA
dmaClass_Reservation	DMA
dmaClass_Scope	DMA
dmaClass_System	DMA
dmaClass_Versionable	Containable

Class ID	Superclass
dmaClass_VersionDescription	Containable
dmaClass_VersionSeries	Containable

4.1.4 dmaClass_ClassDescription

A metadata object describing a class.

Class ID: dmaClass_ClassDescription

Superclass: Metadata

4.1.4.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaClassDescription

4.1.4.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Class Description
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Superclass Description	Yes	-	-	-	Object	Scalar	Class Description
Superclass Property Count	Yes	-	-	Yes	Integer32	Scalar	
Immediate Subclass Descriptions	Yes	-	-	-	Object	List	Class Description
Name Property Index	Yes	-	-	-	Integer32	Scalar	
Property Descriptions	Yes	-	-	Yes	Object	List	Property Description

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Has Include Sub-classes	Yes	-	-	Yes	Boolean	Scalar	
Has Proper Subclass Properties	Yes	-	-	Yes	Boolean	Scalar	
Proper Subclass Property Descriptions	Yes	Yes	Yes	-	Object	List	Property Description

4.1.4.3 Description

There is one Class Description object for each DMA class, including those created dynamically (for example, rows in a search result set). Each class has an instance of the Class Description object, including the Class Description class itself. However, in order to avoid non-terminating recursion, the value of the property `dmaProp_ClassDescription` is NULL for a Class Description of a Class Description. Every Class Description object has the same set of properties defined. (The values of these properties are different for different classes, of course.)

Using the `CreateObject()` method on the Class Description class interface enables you to create an object of that class. The list of Property Description objects in the Class Description object describe the properties of a given class.

4.1.4.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Metadata

- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Metadata
- **Ids {dmaProp_Ids}**
Property Inherited from Metadata
- **Superclass Description {dmaProp_SuperclassDescription}**
Property Attributes: Objects of Class 'Class Description', Property Must Be Supported

The value of this property is the Class Description object of the class that is the immediate superclass of the current class in the DMA property class hierarchy.

The superclass is unique, because the DMA hierarchy supports only single inheritance. This property must have a value except when the current object describes dmaClass_DMA.

- **Superclass Property Count {dmaProp_SuperclassPropertyCount}**

Property Attributes: Integer32, Property Must Be Supported, Value Required

The number of properties in the immediate superclass (and therefore inherited by this class).

This number reflects the ordered count of properties inherited from the superclass.

- **Immediate Subclass Descriptions {dmaProp_ImmediateSubclassDescriptions}**

Property Attributes: List Of Objects of Class 'Class Description', Property Must Be Supported

The value of this property is a list of Class Description objects, one for every class that is an immediate subclass of the current class.

All of the Class Description objects in this list have a Class Description object for the current class as the value of their Superclass Description property.

- **Name Property Index {dmaProp_NamePropertyIndex}**

Property Attributes: Integer32, Property Must Be Supported

The index of the property that serves as the "name" for instances of the class being described.

The value of this property is an index into the list of property description objects of the current Class Description object (see dmaProp_PropertyDe-

scriptions). The Property Description object at the designated index describes the property that is considered by client applications to indicate the "name" of the object instance being described. The purpose of this property is to make it as convenient as possible for DMA clients to access the "name" of the current object instance.

- **Property Descriptions {dmaProp_PropertyDescriptions}**

Property Attributes: List Of Objects of Class 'Property Description', Property Must Be Supported, Value Required

A list of Property Description objects describing this class's properties.

The properties inherited from ancestor classes occur first in the list, and the list ends with the properties introduced by this class, i.e., the "native" properties. The value of the property dmaProp_SuperclassPropertyCount is the number of properties that are inherited.

- **Has Include Subclasses {dmaProp_HasIncludeSubclasses}**

Property Attributes: Boolean, Property Must Be Supported, Value Required

This flag is set if this class supports "include subclasses" queries.

- **Has Proper Subclass Properties {dmaProp_HasProperSubclassProperties}**

Property Attributes: Boolean, Property Must Be Supported, Value Required

This flag is set if this class supports "include descendant properties" queries.

- **Proper Subclass Property Descriptions {dmaProp_ProperSubclassPropertyDescriptions}**

Property Attributes: List Of Objects of Class 'Property Description', Property Must Be Supported, System Generated, Read-Only

This is a list of all properties of all the subclasses of the current class.

This property is used to facilitate "include subclasses" queries. If a property occurs more than once in the descendant subclasses and has different characteristics in some of them, its occurrence in this list resolves how the property should be treated in "include subclasses" queries.

4.1.5 dmaClass_ConfigurationHistory

The root object for a conceptual versioned entity.

Class ID:dmaClass_ConfigurationHistory

Superclass:Containable

4.1.5.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]

4.1.5.2 Properties

Name	Impl. Req'd	Sys-tem Gen'ed	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Configuration History
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Parent	-	Yes	Yes	-	Object	Scalar	Direct Containment Relationship
Parent Container	-	Yes	Yes	-	Object	Scalar	Container
Containers	-	Yes	Yes	-	Object	Enum	Referential Containment Relationship
Versioned Object Class	-	-	-	Yes	Object	Scalar	Class Description
Primary Version Series	Yes	Yes	Yes	-	Object	Scalar	Version Series
All Version Series	Yes	Yes	Yes	-	Object	Enum	Version Series

4.1.5.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Parent {dmaProp_Parent}**
Property Inherited from Containable
- **Parent Container {dmaProp_ParentContainer}**
Property Inherited from Containable
- **Containers {dmaProp_Containers}**
Property Inherited from Containable
- **Versioned Object Class {dmaProp_VersionedObjectClass}**

Property Attributes: Objects of Class 'Class Description', Value Required

Base class supported for this conceptual entity

This is a class description object that specifies the base class that can be versioned using this conceptual object. All objects that participate in a version series of this conceptual object must have this class as a base class. Whether this property can be manipulated after first set or after the first versionable is added is a Document Space policy matter.

- **Primary Version Series {dmaProp_PrimaryVersionSeries}**

Property Attributes: Objects of Class 'Version Series', Property Must Be Supported, System Generated, Read-Only

Primary Version series for this conceptual object.

This is the object that holds the primary version series of the conceptual entity managed by this Configuration History. It is established by creation

of the Version Series object that is specified to represent the primary series attached to this Configuration History.

- **All Version Series {dmaProp_AllVersionSeries}**

Property Attributes: Enum Of Objects of Class 'Version Series', Property Must Be Supported, System Generated, Read-Only

An enumeration of all the version series associated with this conceptual object.

This is a system-derived property. It is an enumeration of those Version Series which comprise the configuration managed under the conceptual entity. For the DMA 1.0 Versioning Model, compliant configurations consist of exactly one Version Series. Therefore, for DMA 1.0, this property contains a single element, which will be the primary version series object.

4.1.6 dmaClass_Containable

The base class for all objects which can be contained, that is, the generic "containee" class.

Class ID: dmaClass_Containable

Superclass: DMA

4.1.6.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]

4.1.6.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Containable
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Parent	-	Yes	Yes	-	Object	Scalar	Direct Containment Relationship
Parent Container	-	Yes	Yes	-	Object	Scalar	Container
Containers	-	Yes	Yes	-	Object	Enum	Referential Containment Relationship

4.1.6.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA

- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Parent {dmaProp_Parent}**

Property Attributes: Objects of Class 'Direct Containment Relationship', System Generated, Read-Only

The Direct Containment Relationship object which identifies the direct containment parent of this object.

This property is the Direct Containment Relationship object which identifies the parent, by direct containment, of this object. The Parent is paired reflectively with the Head property of the Direct Containment Relationship object.

- **Parent Container {dmaProp_ParentContainer}**

Property Attributes: Objects of Class 'Container', System Generated, Read-Only

The object which directly contains this object.

This property is the parent object which directly contains this child object. It is a shortcut property that allows a client to bypass the intermediate DirectContainmentRelationship object. There is no reflective property in the Container object corresponding to the Parent Container property. If this property is noted as searchable in the metadata, it can be used to bypass the relationship object in a query and find parent containers directly.

- **Containers {dmaProp_Containers}**

Property Attributes: Enum Of Objects of Class 'Referential Containment Relationship', System Generated, Read-Only

The enumeration of Referential Containment Relationship objects which identify the containers of this object.

This is an enumeration of Referential Containment Relationship objects which identify the container objects that contain this object, by referential containment. The Containers property is paired reflectively with the Head property of the Referential Containment Relationship object.

4.1.7 dmaClass_Container

An object that can contain other objects either directly or referentially, but cannot have content data of its own.

Class ID: dmaClass_Container

Superclass: Containable

4.1.7.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]

4.1.7.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Container
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Parent	-	Yes	Yes	-	Object	Scalar	Direct Containment Relationship
Parent Container	-	Yes	Yes	-	Object	Scalar	Container
Containers	-	Yes	Yes	-	Object	Enum	Referential Containment Relationship
Children	-	Yes	Yes	-	Object	Enum	Direct Containment Relationship
Containees	-	Yes	Yes	-	Object	Enum	Referential Containment Relationship

4.1.7.3 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Parent {dmaProp_Parent}**
Property Inherited from Containable
- **Parent Container {dmaProp_ParentContainer}**
Property Inherited from Containable
- **Containers {dmaProp_Containers}**
Property Inherited from Containable
- **Children {dmaProp_Children}**

Property Attributes: Enum Of Objects of Class 'Direct Containment Relationship', System Generated, Read-Only

The enumeration of Direct Containment Relationship objects contained within this Superclass.

This is an enumeration of Direct Containment Relationship, each of which represents a direct containment relationship between this object and a child object. Direct containment is a 1:n relationship between a parent object and directly contained child objects. A child object can be directly contained within only one parent container object. Direct containment provides for a strict containment hierarchy with no cycles. The Children property is paired reflectively with the Tail property of the Direct Containment Relationship object.

- **Containees {dmaProp_Containees}**

Property Attributes: Enum Of Objects of Class 'Referential Containment Relationship', System Generated, Read-Only

The enumeration of Referential Containment Relationship objects referentially contained within this container.

This is an enumeration of Referential Containment Relationship objects, each of which represents a referential containment relationship between this object and a containee object. Referential containment is an n:m relationship between a container and referentially contained containee objects. A containee object can be referentially contained within multiple container objects. The Containees property is paired reflectively with the Tail property of the Referential Containment Relationship object..

4.1.8 dmaClass_ContainmentRelationship

Relationship subclass for modeling containment

Class ID: dmaClass_ContainmentRelationship

Superclass: Relationship

4.1.8.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]
- IdmaRelationshipOrdering [Optional]

4.1.8.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Containment Relationship
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Head	Yes	-	-	Yes	Object	Scalar	Containable
Tail	Yes	-	-	Yes	Object	Scalar	Container

4.1.8.3 Description

The base dmaClassContainmentRelationship class models containment within a DMA DocSpace. IT is a subclass of dmaClass_Relationship. It does not add any new properties, but it specializes the properties it inherits from dmaClass_Relationship by constraining the types of objects that can be asso-

ciated to just objects of class dmaClass_Container and dmaClass_Containable.

A document space can set dmaClass_ContainmentRelationship to searchable if it wants to support searching for both direct and referential containment relationships in a single query. If this class is marked non-searchable, DMA clients must search for direct and referential relationships in two separate searches.

4.1.8.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Head {dmaProp_Head}**
Property Inherited from Relationship
- **Tail {dmaProp_Tail}**
Property Inherited from Relationship

4.1.9 dmaClass_ContentElement

Base class for classes used to access document content.

Class ID: dmaClass_ContentElement

Superclass: DMA

4.1.9.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]

4.1.9.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Content Element
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Component Type	Yes	-	-	Yes	String	Scalar	

4.1.9.3 Description

This is the base class for the concrete classes that are used to access document content. Its purpose is to provide a common point for the introduction on the Component Type property.

Implementation of this class requires that dmaClass_Rendition also be implemented.

4.1.9.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA

- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Component Type {dmaProp_ComponentType}**

Property Attributes: String, Property Must Be Supported, Value Required

Content Identifier: A String, used to distinguish multiple content items in a rendition. DMA does not prescribe formats for values of this property, they are intended to be dependent on rendition type.

4.1.10 dmaClass_ContentReference

A content element that represents content that exists outside the control of the document space, but to which the document space maintains a reference.

Class ID: dmaClass_ContentReference

Superclass: Content Element

4.1.10.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]

4.1.10.2 Properties

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip- tion
This	-	Yes	Yes	Yes	Object	Scalar	Content Refer- ence
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Component Type	Yes	-	-	Yes	String	Scalar	
Content Location	Yes	-	-	Yes	String	Scalar	

4.1.10.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Component Type {dmaProp_ComponentType}**
Property Inherited from Content Element
- **Content Location {dmaProp_ContentLocation}**

Property Attributes: String, Property Must Be Supported, Value Required

Specifies, in URL format, the name of the resource which contains the content data.

When delivering new content, the client application must set a value for this property. The document space may validate the syntax of the URL and may even test to see that it refers to an existing resource, but it is required to do neither of these things.

When returning a saved value to subsequent clients, the document space is required only to deliver back the content location string exactly as it was supplied initially, without regard to whether the resource so referenced still exists or whether it contains the same content as at the time the reference was saved. The document space may however deliver back a different string, if it is able, for example:

- to transform the URL into a more globally applicable form
- to apply "link tracking" techniques to discover a new location for content which has been moved. A document space may permit the Content Location property to be modified on an already persistent content reference, but is not required to allow this.

4.1.11 dmaClass_ContentTransfer

A content element that represents content that is, or will be, directly captured and managed by the document space.

Class ID: dmaClass_ContentTransfer

Superclass: Content Element

4.1.11.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaContentTransfer

4.1.11.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Content Transfer
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Component Type	Yes	-	-	Yes	String	Scalar	
Retrieval Name	Yes	-	-	-	String	Scalar	

4.1.11.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Component Type {dmaProp_ComponentType}**
Property Inherited from Content Element
- **Retrieval Name {dmaProp_RetrievalName}**

Property Attributes: String, Property Must Be Supported

Suggested name suffix for the resource into which the document content should be retrieved for editing.

If a value is present in this property, then it specifies the suggested name suffix for the resource into which the document content should be retrieved for editing, with the purpose of allowing inference by extension (for example) as a means of determining the appropriate editing application.

When delivering new content, the client application should set this property to the resource name from which the content was copied, if such a name exists. A syntactically correct URL is required. The document space may save either this entire URL, or it may eliminate everything but the final component of the path (the portion of the string following the last path separator character; this will typically be of the form filename.ext). If the client does not supply a value for this property, and the content is delivered via the `IdmaContentTransfer::SetCaptureResource` method then the document space is at liberty to set a default value by parsing the capture resource name. If such defaulting does take place, then it should become visible only after `ExecuteChange` and `Refresh` have been applied to the parent `DocVersion` object. (`SetCaptureResource` is not permitted to side-effect the value of the Retrieval Name property on the scratchpad object).

During content retrieval the client application may use the value of this property, if present, to construct the name of a resource into which the content is copied. However, it is not obliged to do so.

Document spaces may allow the value of this property to be modified on content element objects which have already been made persistent, but this is not required.

4.1.12 dmaClass_DirectContainmentRelationship

Defines a direct containment relationship

Class ID: dmaClass_DirectContainmentRelationship

Superclass: Containment Relationship

4.1.12.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]
- IdmaRelationshipOrdering [Optional]

4.1.12.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Direct Containment Relationship
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Head	Yes	-	-	Yes	Object	Scalar	Containable
Tail	Yes	-	-	Yes	Object	Scalar	Container

4.1.12.3 Description

The class dmaClass_directContainmentRelationship models direct containment within a DMA DocSpace. It is a subclass of dmaClass_ContainmentRelationship. It introduces no new properties or interfaces. This class and dmaClass_ReferentialContainmentRelationship are needed in the model to al-

low them to have different metadata, in particular searchability. This class can be set to non-searchable by a document space to indicate that searching for direct containment should be done via the shortcut property `dmaProp_ParentContainer` instead.

4.1.12.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Head {dmaProp_Head}**
Property Inherited from Relationship
- **Tail {dmaProp_Tail}**
Property Inherited from Relationship

4.1.13 dmaClass_DMA

The DMA base class.

Class ID: dmaClass_DMA

Superclass: none

4.1.13.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.13.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	DMA
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	

4.1.13.3 Description

The ultimate superclass of all DMA classes. The Object ID, Create Flag, Update Flag and Delete Flag properties introduced here should only be implemented in subclasses which represent independently persistable objects.

4.1.13.4 Property Descriptions

- **OIID {dmaProp_OIID}**

Property Attributes: String, System Generated, Read-Only

This property is the unique instance Id of this particular object.

- **Class Description {dmaProp_ClassDescription}**

Property Attributes: Objects of Class 'Class Description', Property Must Be Supported, Value Required, System Generated, Read-Only

The Class Description object describing this object's class.

This property provides access from this object to the descriptive information about the class of this object.

- **This {dmaProp_This}**

Property Attributes: Objects of Class 'DMA', Value Required, System Generated, Read-Only

A synthetic, object-valued property that has as its implicit value the DMA object itself. It's primary purpose is to enable the expression of relationships among objects in DMA queries and to select candidate objects in query results.

This object-valued property is an exception to the general rules for the by-reference behavior of object-valued properties. When the property is supported by an object, the behavior of

```
rc = pMyObjectProperties -> GetPropValObjectById(&dmaProp_This,  
riid, ppIObjectValue)
```

is equivalent to that of the preferable, always-supported operation

```
rc = pMyObjectProperties -> QueryInterface(riid, ppIObjectValue)
```

Using a DMA object's IdmaProperties interface, the property behaves as if it is always bound and has nothing to do with any persistent object associated with the DMA object. Refreshing and other operations between a scratchpad object and a persistent object have no impact whatsoever on the This property of a DMA object.

[In every class description for a DMA Object inherited from the dmaClass_DMA class, the Required Class of Object restriction is always to the very class being described. In addition, proper subclasses of that class will never occur. This is part of the system-generated quality of dmaProp_This properties.]

- **Create Pending {dmaProp_CreatePending}**

Property Attributes: Boolean, Value Required, System Generated, Read-Only

If set, this flag indicates to ExecuteChange and ExecuteChanges that the current scratchpad object is to be created as a new persistent object.

- **Update Pending {dmaProp_UpdatePending}**

Property Attributes: Boolean, Value Required, System Generated, Read-Only

If set, this flag indicates to `ExecuteChange` and `ExecuteChanges` that the current object is already persistent, and its properties are to be modified.

- **Delete Pending {`dmaProp_DeletePending`}**

Property Attributes: Boolean, Value Required, System Generated, Read-Only

If set, this flag indicates to `ExecuteChange` and `ExecuteChanges` that the current object is already persistent, and is to be deleted.

4.1.14 dmaClass_DocSpace

A Doc Space is a collection of documents and policies.

Class ID: dmaClass_DocSpace

Superclass: DMA

4.1.14.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaObjectFactory [Optional]
- IdmaAuthentication [Optional]
- IdmaBatch [Optional]
- IdmaDocSpace

4.1.14.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Doc Space
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	-	String	Scalar	
Class Descriptions	Yes	-	-	Yes	Object	List	Class Description
Locale Name	Yes	Yes	Yes	Yes	String	Scalar	
Locale Names	Yes	Yes	Yes	Yes	String	List	
Doc Space Id	Yes	Yes	Yes	Yes	ID	Scalar	
Initial Containers	-	Yes	Yes	-	Object	Enum	Container

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Doc Space Capabilities	Yes	Yes	Yes	Yes	Integer32	List	

4.1.14.3 Description

A Doc Space is the source of metadata for all objects except Systems and Document Spaces themselves. Each Document Space publishes the supported objects and how the objects are accessed. There may be class and property definitions that are not meaningful to other Document Spaces. Object valued properties cannot be accessed if the DocSpace object is in an unauthenticated state. Attempts to access an object valued property in an unauthenticated state will yield a DMARC_NOT_AUTHENTICATED return code.

4.1.14.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**

Property Attributes: String, Property Must Be Supported, Value Required

A string containing a user-intelligible name for the class/property/... being described.

- **Descriptive Text {dmaProp_DescriptiveText}**

Property Attributes: String, Property Must Be Supported

Text documenting this object.

- **Class Descriptions {dmaProp_ClassDescriptions}**

Property Attributes: List Of Objects of Class 'Class Description', Property Must Be Supported, Value Required

A list of all class descriptions supported by this document space.

- **Locale Name {dmaProp_LocaleName}**

Property Attributes: String, Property Must Be Supported, Value Required, System Generated, Read-Only

Indicates the name of the Locale this object is currently operating in.

- **Locale Names {dmaProp_LocaleNames}**

Property Attributes: List Of String, Property Must Be Supported, Value Required, System Generated, Read-Only

The list of locales this object supports.

- **Doc Space Id {dmaProp_DocSpaceId}**

Property Attributes: ID, Property Must Be Supported, Value Required, System Generated, Read-Only

The persistent Id that identifies this Document Space.

- **Initial Containers {dmaProp_InitialContainers}**

Property Attributes: Enum Of Objects of Class 'Container', System Generated, Read-Only

An enumeration of folder objects which represent starting points for folder navigation

- **Doc Space Capabilities {dmaProp_DocSpaceCapabilities}**

Property Attributes: List Of Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

List of capabilities for the Doc Space.

Capabilities that a Doc Space supports or provides are listed in this property. An application can discover if a Doc Space provides a feature by checking this property. This allows the application to optimize presentation options to the user without the expense of trying certain operations to see whether they are supported.

4.1.15 dmaClass_DocVersion

A document version is single document stored in a Document Space.

Class ID: dmaClass_DocVersion

Superclass: Versionable

4.1.15.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]
- IdmaVersionable [Optional]

4.1.15.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Doc Version
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Parent	-	Yes	Yes	-	Object	Scalar	Direct Containment Relationship
Parent Container	-	Yes	Yes	-	Object	Scalar	Container
Containers	-	Yes	Yes	-	Object	Enum	Referential Containment Relationship
Version Descriptions	-	Yes	Yes	-	Object	Enum	Version Description
Current Of Series Count	-	Yes	Yes	-	Integer32	Scalar	

Name	Impl. Req'e d	Sys- tem Gen'e d	Read- Only	Value Req'e d	Type	Cardinal- ity	Required Class
Renditions Present	-	Yes	Yes	-	String	List	
Renditions	-	-	-	-	Object	List	Rendition

4.1.15.3 Description

This is the base class for objects representing "documents". It introduces the properties which support the representation of document content (if supported) and inherits properties which support versioning and containment of documents (if supported). A document space may allow actual instances of DocVersion to be created, or it may only allow instances of subclasses of DocVersion.

4.1.15.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Parent {dmaProp_Parent}**
Property Inherited from Containable
- **Parent Container {dmaProp_ParentContainer}**
Property Inherited from Containable
- **Containers {dmaProp_Containers}**
Property Inherited from Containable
- **Version Descriptions {dmaProp_VersionDescriptions}**
Property Inherited from Versionable
- **Current Of Series Count {dmaProp_CurrentOfSeriesCount}**
Property Inherited from Versionable

- **Renditions Present {dmaProp_RenditionsPresent}**

Property Attributes: List Of String, System Generated, Read-Only

List of the types of renditions contained within the DocVersion at the time it was last made persistent.

A system-generated (and read-only) list of Strings which reflect the rendition types of the renditions in the document. This property is static except over a Refresh and does not dynamically track the state of the Renditions list. Therefore, the client application may not in general assume that the number or order of elements in this list correspond to that of the Renditions list.

However, if neither the Rendition list nor any of the rendition objects it contains have been modified since the DocVersion was connected or last refreshed, then it is required that the two lists be synchronized.

- **Renditions {dmaProp_Renditions}**

Property Attributes: List Of Objects of Class 'Rendition'

List of Renditions contained within this DocVersion

4.1.16 dmaClass_Enumeration

A metadata class object describing an enumeration.

Class ID: dmaClass_Enumeration

Superclass: DMA

4.1.16.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.16.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Enumeration
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.16.3 Description

An enumeration of elements of a common data type. An enumeration is different than a list, in that there is an expectation that the number of elements could be large, that the elements being enumerated could be from a docspace that is being updated concurrently, and that the number of elements is not known in advance and can change as the sequence of elements is enumerated due to concurrent updates. Therefore, there is no expectation of reproducibility if the sequence is reset and regenerated.

4.1.16.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA

- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

The data type of the elements in the enumeration.

The allowed values for this property are DMA_DATATYPE_BINARY, DMA_DATATYPE_BOOLEAN, DMA_DATATYPE_DATETIME, DMA_DATATYPE_FLOAT64, DMA_DATATYPE_ID, DMA_DATATYPE_INTEGER32, DMA_DATATYPE_OBJECT, DMA_DATATYPE_STRING, and DMA_DATATYPE_RESULT_ROW.

4.1.17 dmaClass_EnumerationOfObject

An enumeration of object elements.

Class ID: dmaClass_EnumerationOfObject

Superclass: Enumeration

4.1.17.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEnumOfObject

4.1.17.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Enumeration Of Object
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Required Class	Yes	-	-	-	Object	Scalar	Class Description

4.1.17.3 Description

An enumeration of a sequence of objects is useful for various purposes including enumerating the contents of a container, enumerating the containers of a containee, and enumerating the result rows of a query.

4.1.17.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA

- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**
Property Inherited from Enumeration
- **Required Class {dmaProp_RequiredClass}**

Property Attributes: Objects of Class 'Class Description', Property Must Be Supported

Specifies the class to which all objects delivered by the enumeration will conform.

4.1.18 dmaClass_List

A list of elements of a common data type.

Class ID: dmaClass_List

Superclass: DMA

4.1.18.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaList
- IdmaEditList [Optional]

4.1.18.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	List
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Requires Unique Elements	Yes	Yes	Yes	Yes	Boolean	Scalar	
Is Order Preserving	Yes	Yes	Yes	Yes	Boolean	Scalar	
Maximum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	
Minimum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.18.3 Description

The List object contains an ordered, possibly empty sequence of elements having a common data type. Modifiable lists may have elements inserted or

deleted at any position in the list. List elements are retrievable by index (zero origin). There are subclasses for lists of each data type.

4.1.18.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

The data type of the elements in the list.

The DMA data types (in order) are: DmaBinary, DmaBoolean, DmaDateTime, DmaFloat64, DmaId, DmaInteger32, DmaObject, DmaOIID, and DmaString.

The defined symbols for dmaProp_DataType property values for Element Data Type are DMA_DATATYPE_BINARY, DMA_DATATYPE_BOOLEAN, DMA_DATATYPE_DATETIME, DMA_DATATYPE_FLOAT64, DMA_DATATYPE_ID, DMA_DATATYPE_INTEGER32, DMA_DATATYPE_OBJECT, DMA_DATATYPE_STRING, and DMA_DATATYPE_STRING for specifying the respective data types.

- **Requires Unique Elements {dmaProp_RequiresUniqueElements}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

When true, the values of the list must be unique. Attempting to insert a duplicate value can result in the immediate or deferred failure code DMARC_NOT_UNIQUE. Generally, it is necessary to first delete any unwanted entry that would be duplicated.

When false, there is no requirement for uniqueness of the values.

- **Is Order Preserving {dmaProp_IsOrderPreserving}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

When true, the order of the elements of the list is preserved. When the list is made persistent in some way, the elements will be returned in precisely the same sequence for all DMA List instances bound before any intervening change occurs.

When false, the list elements may be returned in a different order for future bindings of the persistent object, even though there has been no intervening change in the DMA object. Typically, the persistent implementation has some ordering mechanism that may lead to return of values in a different and not necessarily predictable sequence.

- **Maximum Elements {dmaProp_MaximumElements}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

There may be no more elements in the list than the number specified by this property.

The value DMA_NO_MAXIMUM (-1) is used to indicate that the list has no fixed maximum on the number of elements that it may carry.

Attempting to insert more elements than specified in this property can result in the immediate or deferred failure code DMA_LIST_BOUNDS_ERROR.

- **Minimum Elements {dmaProp_MinimumElements}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

There may be no fewer elements in the list than the number specified by this property.

The value 0 is used to indicate that the list is permitted to be empty. A list can be restricted to a specific size by having the same values for dmaProp_MinimumElements and dmaProp_MaximumElements.

Attempting to employ the list when it has fewer elements than specified by this property can lead to an (usually deferred) failure code DMA_LIST_BOUNDS_ERROR. It is desirable that the list implementation tolerate deletions that result in fewer than the minimum number of elements, so the client has an opportunity to complete the list later.

4.1.19 dmaClass_ListOfBinary

A list of binary elements.

Class ID: dmaClass_ListOfBinary

Superclass: List

4.1.19.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaList
- IdmaListOfBinary
- IdmaEditList [Optional]
- IdmaEditListOfBinary [Optional]

4.1.19.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	List Of Binary
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Requires Unique Elements	Yes	Yes	Yes	Yes	Boolean	Scalar	
Is Order Preserving	Yes	Yes	Yes	Yes	Boolean	Scalar	
Maximum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	
Minimum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.19.3 Description

A subclass of dmaClass_List. The List object contains an ordered, possibly empty, set of elements having a binary data type. Modifiable lists may have elements inserted or deleted at any position in the list. List elements are retrievable by index (zero origin).

4.1.19.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**
Property Inherited from List
- **Requires Unique Elements {dmaProp_RequiresUniqueElements}**
Property Inherited from List
- **Is Order Preserving {dmaProp_IsOrderPreserving}**
Property Inherited from List
- **Maximum Elements {dmaProp_MaximumElements}**
Property Inherited from List
- **Minimum Elements {dmaProp_MinimumElements}**
Property Inherited from List

4.1.20 dmaClass_ListOfBoolean

A list of boolean elements.

Class ID: dmaClass_ListOfBoolean

Superclass: List

4.1.20.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaList
- IdmaListOfBoolean
- IdmaEditList [Optional]
- IdmaEditListOfBoolean [Optional]

4.1.20.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	List Of Boolean
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Requires Unique Elements	Yes	Yes	Yes	Yes	Boolean	Scalar	
Is Order Preserving	Yes	Yes	Yes	Yes	Boolean	Scalar	
Maximum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	
Minimum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.20.3 Description

A subclass of dmaClass_List. The List object contains an ordered, possibly empty, set of elements having a boolean data type. Modifiable lists may have elements inserted or deleted at any position in the list. List elements are retrievable by index (zero origin).

4.1.20.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**
Property Inherited from List
- **Requires Unique Elements {dmaProp_RequiresUniqueElements}**
Property Inherited from List
- **Is Order Preserving {dmaProp_IsOrderPreserving}**
Property Inherited from List
- **Maximum Elements {dmaProp_MaximumElements}**
Property Inherited from List
- **Minimum Elements {dmaProp_MinimumElements}**
Property Inherited from List

4.1.21 dmaClass_ListOfDateTime

A list of Date Time elements.

Class ID: dmaClass_ListOfDateTime

Superclass: List

4.1.21.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaList
- IdmaListOfDateTime
- IdmaEditList [Optional]
- IdmaEditListOfDateTime [Optional]

4.1.21.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	List Of Date Time
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Requires Unique Elements	Yes	Yes	Yes	Yes	Boolean	Scalar	
Is Order Preserving	Yes	Yes	Yes	Yes	Boolean	Scalar	
Maximum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	
Minimum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.21.3 Description

A subclass of dmaClass_List. The List object contains an ordered, possibly empty, set of elements having a Date Time data type. Modifiable lists may have elements inserted or deleted at any position in the list. List elements are retrievable by index (zero origin).

4.1.21.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**
Property Inherited from List
- **Requires Unique Elements {dmaProp_RequiresUniqueElements}**
Property Inherited from List
- **Is Order Preserving {dmaProp_IsOrderPreserving}**
Property Inherited from List
- **Maximum Elements {dmaProp_MaximumElements}**
Property Inherited from List
- **Minimum Elements {dmaProp_MinimumElements}**
Property Inherited from List

4.1.22 dmaClass_ListOfFloat64

A list of float elements.

Class ID: dmaClass_ListOfFloat64

Superclass: List

4.1.22.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaList
- IdmaListOfFloat64
- IdmaEditList [Optional]
- IdmaEditListOfFloat64 [Optional]

4.1.22.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	List Of Float
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Requires Unique Elements	Yes	Yes	Yes	Yes	Boolean	Scalar	
Is Order Preserving	Yes	Yes	Yes	Yes	Boolean	Scalar	
Maximum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	
Minimum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.22.3 Description

A subclass of dmaClass_List. The List object contains an ordered, possibly empty, set of elements having a float64 data type. Modifiable lists may have elements inserted or deleted at any position in the list. List elements are retrievable by index (zero origin).

4.1.22.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**
Property Inherited from List
- **Requires Unique Elements {dmaProp_RequiresUniqueElements}**
Property Inherited from List
- **Is Order Preserving {dmaProp_IsOrderPreserving}**
Property Inherited from List
- **Maximum Elements {dmaProp_MaximumElements}**
Property Inherited from List
- **Minimum Elements {dmaProp_MinimumElements}**
Property Inherited from List

4.1.23 dmaClass_ListOfId

A list of ID elements.

Class ID: dmaClass_ListOfId

Superclass: List

4.1.23.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaList
- IdmaListOfId
- IdmaEditList [Optional]
- IdmaEditListOfId [Optional]

4.1.23.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	List Of ID
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Requires Unique Elements	Yes	Yes	Yes	Yes	Boolean	Scalar	
Is Order Preserving	Yes	Yes	Yes	Yes	Boolean	Scalar	
Maximum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	
Minimum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.23.3 Description

A subclass of dmaClass_List. The List object contains an ordered, possibly empty, set of elements having a ID data type. Modifiable lists may have elements inserted or deleted at any position in the list. List elements are retrievable by index (zero origin).

4.1.23.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**
Property Inherited from List
- **Requires Unique Elements {dmaProp_RequiresUniqueElements}**
Property Inherited from List
- **Is Order Preserving {dmaProp_IsOrderPreserving}**
Property Inherited from List
- **Maximum Elements {dmaProp_MaximumElements}**
Property Inherited from List
- **Minimum Elements {dmaProp_MinimumElements}**
Property Inherited from List

4.1.24 dmaClass_ListOfInteger32

A list of integer elements.

Class ID: dmaClass_ListOfInteger32

Superclass: List

4.1.24.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaList
- IdmaListOfInteger32
- IdmaEditList [Optional]
- IdmaEditListOfInteger32 [Optional]

4.1.24.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	List Of Integer
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Requires Unique Elements	Yes	Yes	Yes	Yes	Boolean	Scalar	
Is Order Preserving	Yes	Yes	Yes	Yes	Boolean	Scalar	
Maximum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	
Minimum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.24.3 Description

A subclass of dmaClass_List. The List object contains an ordered, possibly empty, set of elements having an Integer32 data type. Modifiable lists may have elements inserted or deleted at any position in the list. List elements are retrievable by index (zero origin).

4.1.24.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**
Property Inherited from List
- **Requires Unique Elements {dmaProp_RequiresUniqueElements}**
Property Inherited from List
- **Is Order Preserving {dmaProp_IsOrderPreserving}**
Property Inherited from List
- **Maximum Elements {dmaProp_MaximumElements}**
Property Inherited from List
- **Minimum Elements {dmaProp_MinimumElements}**
Property Inherited from List

4.1.25 dmaClass_ListOfObject

A list of object elements.

Class ID: dmaClass_ListOfObject

Superclass: List

4.1.25.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaList
- IdmaListOfObject
- IdmaEditList [Optional]
- IdmaEditListOfObject [Optional]

4.1.25.2 Properties

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip- tion
This	-	Yes	Yes	Yes	Object	Scalar	List Of Object
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Requires Unique Elements	Yes	Yes	Yes	Yes	Boolean	Scalar	
Is Order Preserving	Yes	Yes	Yes	Yes	Boolean	Scalar	
Maximum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	
Minimum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Required Class	Yes	Yes	Yes	-	Object	Scalar	Class Description

4.1.25.3 Description

A subclass of dmaClass_List. The List object contains an ordered, possibly empty, set of object elements, all of the same class. Modifiable lists may have elements inserted or deleted at any position in the list. List elements are retrievable by index (zero origin).

4.1.25.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**
Property Inherited from List
- **Requires Unique Elements {dmaProp_RequiresUniqueElements}**
Property Inherited from List
- **Is Order Preserving {dmaProp_IsOrderPreserving}**
Property Inherited from List
- **Maximum Elements {dmaProp_MaximumElements}**
Property Inherited from List
- **Minimum Elements {dmaProp_MinimumElements}**
Property Inherited from List

- **Required Class {dmaProp_RequiredClass}**

Property Attributes: Objects of Class 'Class Description', Property Must Be Supported, System Generated, Read-Only

Specifies the class to which all the object elements must conform. If there is no particular requirement, this property may be null.

4.1.26 dmaClass_ListOfString

A list of string elements.

Class ID: dmaClass_ListOfString

Superclass: List

4.1.26.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaList
- IdmaListOfString
- IdmaEditList [Optional]
- IdmaEditListOfString [Optional]

4.1.26.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	List Of String
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Requires Unique Elements	Yes	Yes	Yes	Yes	Boolean	Scalar	
Is Order Preserving	Yes	Yes	Yes	Yes	Boolean	Scalar	
Maximum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	
Minimum Elements	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.26.3 Description

A subclass of dmaClass_List. The List object contains an ordered, possibly empty, set of elements having a string data type. Modifiable lists may have elements inserted or deleted at any position in the list. List elements are retrievable by index (zero origin).

4.1.26.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**
Property Inherited from List
- **Requires Unique Elements {dmaProp_RequiresUniqueElements}**
Property Inherited from List
- **Is Order Preserving {dmaProp_IsOrderPreserving}**
Property Inherited from List
- **Maximum Elements {dmaProp_MaximumElements}**
Property Inherited from List
- **Minimum Elements {dmaProp_MinimumElements}**
Property Inherited from List

4.1.27 dmaClass_Metadata

The base for the metadata classes that describe classes, properties and operators.

Class ID: dmaClass_Metadata

Superclass: DMA

4.1.27.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.27.2 Properties

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip-tion
This	-	Yes	Yes	Yes	Object	Scalar	Metadata
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	

4.1.27.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA

- **Update Pending {dmaProp_UpdatePending}**

Property Inherited from DMA

- **Delete Pending {dmaProp_DeletePending}**

Property Inherited from DMA

- **Display Name {dmaProp_DisplayName}**

Property Attributes: String, Property Must Be Supported, Value Required

A string containing a user-intelligible name for the class/property/... being described.

- **Descriptive Text {dmaProp_DescriptiveText}**

Property Attributes: String, Property Must Be Supported, Value Required

Text documenting this object.

- **Ids {dmaProp_Ids}**

Property Attributes: List Of ID, Property Must Be Supported, Value Required

A list of the identifiers that serve as the name(s) of the entity (class, property or operator) being described.

The list must contain at least one element and may contain more than one if aliases have been established for the purpose of enabling metadata unification across multiple document spaces. If the entity described is one specified by DMA, the well-known identifier for that entity must be present in the list.

4.1.28 dmaClass_PropertyDescription

A metadata class object describing a property.

Class ID: dmaClass_PropertyDescription

Superclass: Metadata

4.1.28.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.28.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Property Description
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Cardinality	Yes	-	-	Yes	Integer32	Scalar	
Is Selectable	Yes	-	-	Yes	Boolean	Scalar	
Is Searchable	Yes	-	-	Yes	Boolean	Scalar	
Is Orderable	Yes	-	-	Yes	Boolean	Scalar	
Query Operator Descriptions	Yes	-	-	-	Object	List	Query Operator Description
Is System Generated	Yes	-	-	Yes	Boolean	Scalar	
Is Read Only	Yes	-	-	Yes	Boolean	Scalar	

Is Value Required	Yes	-	-	Yes	Boolean	Scalar	
Is Hidden	Yes	-	-	Yes	Boolean	Scalar	

4.1.28.3 Description

Each instance of a Property Description object describes a specific property for a single class, and is an element of the `dmaProp_PropertyList` property of a Class Description object for that class. There is information provided in this object to enable applications to request appropriate data from users (for example, length, min/max, selection lists).

4.1.28.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Metadata
- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Metadata
- **Ids {dmaProp_Ids}**
Property Inherited from Metadata
- **Data Type {dmaProp_DataType}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

The data type of the property, an enumeration.

The DMA data types (in order) are: `DmaBinary`, `DmaBoolean`, `DmaDateTime`, `DmaFloat64`, `DmaId`, `DmaInteger32`, `DmaObject`, `DmaOIID`, and `DmaString`.

The defined symbols for dmaProp_DataType property values for Element Data Type are DMA_DATATYPE_BINARY, DMA_DATATYPE_BOOLEAN, DMA_DATATYPE_DATETIME, DMA_DATATYPE_FLOAT64, DMA_DATATYPE_ID, DMA_DATATYPE_INTEGER32, DMA_DATATYPE_OBJECT, DMA_DATATYPE_STRING, and DMA_DATATYPE_STRING for specifying the respective data types.

- **Cardinality {dmaProp_Cardinality}**

Property Attributes: Integer32, Property Must Be Supported, Value Required

The value of this property indicates the cardinality of the property being described: scalar, list or enumeration.

- **Is Selectable {dmaProp_IsSelectable}**

Property Attributes: Boolean, Property Must Be Supported, Value Required

A boolean specifying whether the class or property can be selected in a query result set.

- **Is Searchable {dmaProp_IsSearchable}**

Property Attributes: Boolean, Property Must Be Supported, Value Required

A boolean specifying whether the class or property can be used as part of the search criteria.

- **Is Orderable {dmaProp_IsOrderable}**

Property Attributes: Boolean, Property Must Be Supported, Value Required

If True, this property can appear in the Orderings list of a query.

- **Query Operator Descriptions {dmaProp_QueryOperatorDescriptions}**

Property Attributes: List Of Objects of Class 'Query Operator Description', Property Must Be Supported

A list of query operator descriptions that this property might legally be an operand of.

- **Is System Generated {dmaProp_IsSystemGenerated}**

Property Attributes: Boolean, Property Must Be Supported, Value Required

If TRUE, indicates that this property is generated and set by the DMA system, usually as a result of some other object's existence.

- **Is Read Only {dmaProp_IsReadOnly}**

Property Attributes: Boolean, Property Must Be Supported, Value Required

For a singleton property, a setting of TRUE implies that the methods of `IdmaEditProperties` will fail when applied to this property.

For a list property, a setting of TRUE implies that a list object obtained from this property will not be modifiable, i.e. that the methods of `IdmaEditListOfXXXX` will fail. The methods of `IdmaEditProperties` always fail on a list property, regardless of the value of this metaproperty.

For an enumeration property, the value of this metaproperty is irrelevant. The methods of `IdmaEditProperties` always fail on an enumeration property and there are no modification methods on an enumeration object.

- **Is Value Required {dmaProp_IsValueRequired}**

Property Attributes: Boolean, Property Must Be Supported, Value Required

If TRUE, indicates that every instance of the described property must have a value.

For a list or enumeration property, this means that the list or enumeration must have at least one element. (A list or enumeration object is always present in the property, irrespective of the setting of Is Value Required.)

- **Is Hidden {dmaProp_IsHidden}**

Property Attributes: Boolean, Property Must Be Supported, Value Required

Hidden property for display to administrative users

If TRUE, denotes a "system" property that should not be displayed to non-administrative users.

4.1.29 dmaClass_PropertyDescriptionBinary

A metadata class object describing a binary property.

Class ID: dmaClass_PropertyDescriptionBinary

Superclass: Property Description

4.1.29.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.29.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Property Description Binary
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Cardinality	Yes	-	-	Yes	Integer32	Scalar	
Is Selectable	Yes	-	-	Yes	Boolean	Scalar	
Is Searchable	Yes	-	-	Yes	Boolean	Scalar	
Is Orderable	Yes	-	-	Yes	Boolean	Scalar	
Query Operator Descriptions	Yes	-	-	-	Object	List	Query Operator Description
Is System Generated	Yes	-	-	Yes	Boolean	Scalar	
Is Read Only	Yes	-	-	Yes	Boolean	Scalar	

Is Value Required	Yes	-	-	Yes	Boolean	Scalar	
Is Hidden	Yes	-	-	Yes	Boolean	Scalar	
Property Default Binary	Yes	-	-	-	Binary	Scalar	
Property Selection Binary	Yes	-	-	-	Binary	List	
Maximum Length Binary	Yes	-	-	Yes	Integer32	Scalar	

4.1.29.3 Description

The current object is an element in the Property List of a Class Description object. The current object describes a single property of the class being described by the Class Description object. The base datatype of the property being described is binary.

4.1.29.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Metadata
- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Metadata
- **Ids {dmaProp_Ids}**
Property Inherited from Metadata
- **Data Type {dmaProp_DataType}**
Property Inherited from Property Description
- **Cardinality {dmaProp_Cardinality}**
Property Inherited from Property Description

- **Is Selectable {dmaProp_IsSelectable}**
Property Inherited from Property Description
- **Is Searchable {dmaProp_IsSearchable}**
Property Inherited from Property Description
- **Is Orderable {dmaProp_IsOrderable}**
Property Inherited from Property Description
- **Query Operator Descriptions {dmaProp_QueryOperatorDescriptions}**
Property Inherited from Property Description
- **Is System Generated {dmaProp_IsSystemGenerated}**
Property Inherited from Property Description
- **Is Read Only {dmaProp_IsReadOnly}**
Property Inherited from Property Description
- **Is Value Required {dmaProp_IsValueRequired}**
Property Inherited from Property Description
- **Is Hidden {dmaProp_IsHidden}**
Property Inherited from Property Description
- **Property Default Binary {dmaProp_PropertyDefaultBinary}**

Property Attributes: Binary, Property Must Be Supported

The default binary value for this property.
- **Property Selection Binary {dmaProp_PropertySelectionsBinary}**

Property Attributes: List Of Binary, Property Must Be Supported

The list of binary values that may be assigned to this property. If a selection list is provided, only a value existing in the list can be assigned to this property.
- **Maximum Length Binary {dmaProp_MaximumLengthBinary}**

Property Attributes: Integer32, Property Must Be Supported, Value Required

The maximum length, in bytes, of a value for the described property. The value -1 (DMA_NO_MAXIMUM) indicates no restriction on length.

4.1.30 dmaClass_PropertyDescriptionBoolean

A metadata class object describing a boolean property.

Class ID: dmaClass_PropertyDescriptionBoolean

Superclass: Property Description

4.1.30.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.30.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Property Description Boolean
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Cardinality	Yes	-	-	Yes	Integer32	Scalar	
Is Selectable	Yes	-	-	Yes	Boolean	Scalar	
Is Searchable	Yes	-	-	Yes	Boolean	Scalar	
Is Orderable	Yes	-	-	Yes	Boolean	Scalar	
Query Operator Descriptions	Yes	-	-	-	Object	List	Query Operator Description
Is System Generated	Yes	-	-	Yes	Boolean	Scalar	

Is Read Only	Yes	-	-	Yes	Boolean	Scalar	
Is Value Required	Yes	-	-	Yes	Boolean	Scalar	
Is Hidden	Yes	-	-	Yes	Boolean	Scalar	
Property Default Boolean	Yes	-	-	-	Boolean	Scalar	
Property Selections Boolean	Yes	-	-	-	Boolean	List	

4.1.30.3 Description

The current object is an element in the Property List of a Class Description object. The current object describes a single property of the class being described by the Class Description object. The base datatype of the property being described is boolean.

4.1.30.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Property Description
- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Property Description
- **Ids {dmaProp_Ids}**
Property Inherited from Property Description
- **Data Type {dmaProp_DataType}**
Property Inherited from Property Description
- **Cardinality {dmaProp_Cardinality}**
Property Inherited from Property Description

- **Is Selectable {dmaProp_IsSelectable}**
Property Inherited from Property Description
- **Is Searchable {dmaProp_IsSearchable}**
Property Inherited from Property Description
- **Is Orderable {dmaProp_IsOrderable}**
Property Inherited from Property Description
- **Query Operator Descriptions {dmaProp_QueryOperatorDescriptions}**
Property Inherited from Property Description
- **Is System Generated {dmaProp_IsSystemGenerated}**
Property Inherited from Property Description
- **Is Read Only {dmaProp_IsReadOnly}**
Property Inherited from Property Description
- **Is Value Required {dmaProp_IsValueRequired}**
Property Inherited from Property Description
- **Is Hidden {dmaProp_IsHidden}**
Property Inherited from Property Description
- **Property Default Boolean {dmaProp_PropertyDefaultBoolean}**

Property Attributes: Boolean, Property Must Be Supported

The default boolean value for this property.
- **Property Selections Boolean {dmaProp_PropertySelectionsBoolean}**

Property Attributes: List Of Boolean, Property Must Be Supported

The list of boolean values that may be assigned to this property. If a selection list is provided, only a value existing in the list can be assigned to this property.

4.1.31 dmaClass_PropertyDescriptionDateTime

A metadata class object describing a Date Time property.

Class ID: dmaClass_PropertyDescriptionDateTime

Superclass: Property Description

4.1.31.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.31.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Property Description Date Time
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Cardinality	Yes	-	-	Yes	Integer32	Scalar	
Is Selectable	Yes	-	-	Yes	Boolean	Scalar	
Is Searchable	Yes	-	-	Yes	Boolean	Scalar	
Is Orderable	Yes	-	-	Yes	Boolean	Scalar	
Query Operator Descriptions	Yes	-	-	-	Object	List	Query Operator Description
Is System Generated	Yes	-	-	Yes	Boolean	Scalar	

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
Is Read Only	Yes	-	-	Yes	Boolean	Scalar	
Is Value Required	Yes	-	-	Yes	Boolean	Scalar	
Is Hidden	Yes	-	-	Yes	Boolean	Scalar	
Property Default Date Time	Yes	-	-	-	Date Time	Scalar	
Property Selections Date Time	Yes	-	-	-	Date Time	List	
Property Minimum Date Time	Yes	-	-	-	Date Time	Scalar	
Property Maximum Date Time	Yes	-	-	-	Date Time	Scalar	

4.1.31.3 Description

The current object is an element in the Property List of a Class Description object. The current object describes a single property of the class being described by the Class Description object. The base datatype of the property being described is Date Time.

4.1.31.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Property Description
- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Property Description

- **Ids {dmaProp_Ids}**
Property Inherited from Property Description
- **Data Type {dmaProp_DataType}**
Property Inherited from Property Description
- **Cardinality {dmaProp_Cardinality}**
Property Inherited from Property Description
- **Is Selectable {dmaProp_IsSelectable}**
Property Inherited from Property Description
- **Is Searchable {dmaProp_IsSearchable}**
Property Inherited from Property Description
- **Is Orderable {dmaProp_IsOrderable}**
Property Inherited from Property Description
- **Query Operator Descriptions {dmaProp_QueryOperatorDescriptions}**
Property Inherited from Property Description
- **Is System Generated {dmaProp_IsSystemGenerated}**
Property Inherited from Property Description
- **Is Read Only {dmaProp_IsReadOnly}**
Property Inherited from Property Description
- **Is Value Required {dmaProp_IsValueRequired}**
Property Inherited from Property Description
- **Is Hidden {dmaProp_IsHidden}**
Property Inherited from Property Description
- **Property Default Date Time {dmaProp_PropertyDefaultDateTime}**

Property Attributes: Date Time, Property Must Be Supported

The default Date Time value for this property.
- **Property Selections Date Time {dmaProp_PropertySelectionsDateTime}**

Property Attributes: List Of Date Time, Property Must Be Supported

The list of Date Time values that may be assigned to this property.

If a selection list is provided, only a value existing in the list can be assigned to this property.
- **Property Minimum Date Time {dmaProp_PropertyMinimumDateTime}**

Property Attributes: Date Time, Property Must Be Supported

The minimum value acceptable for this property.

- **Property Maximum Date Time {dmaProp_PropertyMaximumDateTime}**

Property Attributes: Date Time, Property Must Be Supported

The maximum value acceptable for this property.

4.1.32 dmaClass_PropertyDescriptionFloat64

A metadata class object describing a float property.

Class ID: dmaClass_PropertyDescriptionFloat64

Superclass: Property Description

4.1.32.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.32.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Property Description Float64
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Cardinality	Yes	-	-	Yes	Integer32	Scalar	
Is Selectable	Yes	-	-	Yes	Boolean	Scalar	
Is Searchable	Yes	-	-	Yes	Boolean	Scalar	
Is Orderable	Yes	-	-	Yes	Boolean	Scalar	
Query Operator Descriptions	Yes	-	-	-	Object	List	Query Operator Description
Is System Generated	Yes	-	-	Yes	Boolean	Scalar	

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Is Read Only	Yes	-	-	Yes	Boolean	Scalar	
Is Value Required	Yes	-	-	Yes	Boolean	Scalar	
Is Hidden	Yes	-	-	Yes	Boolean	Scalar	
Property Default Float64	Yes	-	-	-	Float64	Scalar	
Property Selections Float64	Yes	-	-	-	Float64	List	
Property Minimum Float64	Yes	-	-	-	Float64	Scalar	
Property Maximum Float64	Yes	-	-	-	Float64	Scalar	

4.1.32.3 Description

The current object is an element in the Property List of a Class Description object. The current object describes a single property of the class being described by the Class Description object. The base datatype of the property being described is Float64.

4.1.32.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Property Description
- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Property Description

- **Ids {dmaProp_Ids}**
Property Inherited from Property Description
- **Data Type {dmaProp_DataType}**
Property Inherited from Property Description
- **Cardinality {dmaProp_Cardinality}**
Property Inherited from Property Description
- **Is Selectable {dmaProp_IsSelectable}**
Property Inherited from Property Description
- **Is Searchable {dmaProp_IsSearchable}**
Property Inherited from Property Description
- **Is Orderable {dmaProp_IsOrderable}**
Property Inherited from Property Description
- **Query Operator Descriptions {dmaProp_QueryOperatorDescriptions}**
Property Inherited from Property Description
- **Is System Generated {dmaProp_IsSystemGenerated}**
Property Inherited from Property Description
- **Is Read Only {dmaProp_IsReadOnly}**
Property Inherited from Property Description
- **Is Value Required {dmaProp_IsValueRequired}**
Property Inherited from Property Description
- **Is Hidden {dmaProp_IsHidden}**
Property Inherited from Property Description
- **Property Default Float64 {dmaProp_PropertyDefaultFloat64}**

Property Attributes: Float64, Property Must Be Supported

The default Float value for this property.
- **Property Selections Float64 {dmaProp_PropertySelectionsFloat64}**

Property Attributes: List Of Float64, Property Must Be Supported

The list of float values that may be assigned to this property.

If a selection list is provided, only a value existing in the list can be assigned to this property.
- **Property Minimum Float64 {dmaProp_PropertyMinimumFloat64}**

Property Attributes: Float64, Property Must Be Supported. The minimum value acceptable for this property.

- **Property Maximum Float64 {dmaProp_PropertyMaximumFloat64}**

Property Attributes: Float64, Property Must Be Supported. The maximum value acceptable for this property.

4.1.33 dmaClass_PropertyDescriptionId

A metadata class object describing an ID property.

Class ID: dmaClass_PropertyDescriptionId

Superclass: Property Description

4.1.33.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.33.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Property Description Id
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Cardinality	Yes	-	-	Yes	Integer32	Scalar	
Is Selectable	Yes	-	-	Yes	Boolean	Scalar	
Is Searchable	Yes	-	-	Yes	Boolean	Scalar	
Is Orderable	Yes	-	-	Yes	Boolean	Scalar	
Query Operator Descriptions	Yes	-	-	-	Object	List	Query Operator Description
Is System Generated	Yes	-	-	Yes	Boolean	Scalar	
Is Read Only	Yes	-	-	Yes	Boolean	Scalar	

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Is Value Required	Yes	-	-	Yes	Boolean	Scalar	
Is Hidden	Yes	-	-	Yes	Boolean	Scalar	
Property Default Id	Yes	-	-	-	ID	Scalar	
Property Selections Id	Yes	-	-	-	ID	List	

4.1.33.3 Description

The current object is an element in the Property List of a Class Description object. The current object describes a single property of the class being described by the Class Description object. The base datatype of the property being described is ID.

4.1.33.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Property Description
- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Property Description
- **Ids {dmaProp_Ids}**
Property Inherited from Property Description
- **Data Type {dmaProp_DataType}**
Property Inherited from Property Description

- **Cardinality {dmaProp_Cardinality}**
Property Inherited from Property Description
- **Is Selectable {dmaProp_IsSelectable}**
Property Inherited from Property Description
- **Is Searchable {dmaProp_IsSearchable}**
Property Inherited from Property Description
- **Is Orderable {dmaProp_IsOrderable}**
Property Inherited from Property Description
- **Query Operator Descriptions {dmaProp_QueryOperatorDescriptions}**
Property Inherited from Property Description
- **Is System Generated {dmaProp_IsSystemGenerated}**
Property Inherited from Property Description
- **Is Read Only {dmaProp_IsReadOnly}**
Property Inherited from Property Description
- **Is Value Required {dmaProp_IsValueRequired}**
Property Inherited from Property Description
- **Is Hidden {dmaProp_IsHidden}**
Property Inherited from Property Description
- **Property Default Id {dmaProp_PropertyDefaultId}**

Property Attributes: ID, Property Must Be Supported. The default Id value for this property.

- **Property Selections Id {dmaProp_PropertySelectionsId}**

Property Attributes: List Of ID, Property Must Be Supported

The list of Id values that may be assigned to this property. If a selection list is provided, only a value existing in the list can be assigned to this property.

4.1.34 dmaClass_PropertyDescriptionInteger32

A metadata class object describing an integer property.

Class ID: dmaClass_PropertyDescriptionInteger32

Superclass: Property Description

4.1.34.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.34.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Property Description Integer32
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Cardinality	Yes	-	-	Yes	Integer32	Scalar	
Is Selectable	Yes	-	-	Yes	Boolean	Scalar	
Is Searchable	Yes	-	-	Yes	Boolean	Scalar	
Is Orderable	Yes	-	-	Yes	Boolean	Scalar	
Query Operator Descriptions	Yes	-	-	-	Object	List	Query Operator Description
Is System Generated	Yes	-	-	Yes	Boolean	Scalar	

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
Is Read Only	Yes	-	-	Yes	Boolean	Scalar	
Is Value Required	Yes	-	-	Yes	Boolean	Scalar	
Is Hidden	Yes	-	-	Yes	Boolean	Scalar	
Property Default Integer32	Yes	-	-	-	Integer32	Scalar	
Property Selections Integer32	Yes	-	-	-	Integer32	List	
Property Minimum Integer32	Yes	-	-	-	Integer32	Scalar	
Property Maximum Integer32	Yes	-	-	-	Integer32	Scalar	

4.1.34.3 Description

The current object is an element in the Property List of a Class Description object. The current object describes a single property of the class being described by the Class Description object. The base datatype of the property being described is Integer32.

4.1.34.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Property Description
- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Property Description

- **Ids {dmaProp_Ids}**
Property Inherited from Property Description
- **Data Type {dmaProp_DataType}**
Property Inherited from Property Description
- **Cardinality {dmaProp_Cardinality}**
Property Inherited from Property Description
- **Is Selectable {dmaProp_IsSelectable}**
Property Inherited from Property Description
- **Is Searchable {dmaProp_IsSearchable}**
Property Inherited from Property Description
- **Is Orderable {dmaProp_IsOrderable}**
Property Inherited from Property Description
- **Query Operator Descriptions {dmaProp_QueryOperatorDescriptions}**
Property Inherited from Property Description
- **Is System Generated {dmaProp_IsSystemGenerated}**
Property Inherited from Property Description
- **Is Read Only {dmaProp_IsReadOnly}**
Property Inherited from Property Description
- **Is Value Required {dmaProp_IsValueRequired}**
Property Inherited from Property Description
- **Is Hidden {dmaProp_IsHidden}**
Property Inherited from Property Description
- **Property Default Integer32 {dmaProp_PropertyDefaultInteger32}**

Property Attributes: Integer32, Property Must Be Supported

The default integer value for this property.
- **Property Selections Integer32 {dmaProp_PropertySelectionsInteger32}**

Property Attributes: List Of Integer32, Property Must Be Supported

The list of integer values that may be assigned to this property.

If a selection list is provided, only a value existing in the list can be assigned to this property.
- **Property Minimum Integer32 {dmaProp_PropertyMinimumInteger32}**

Property Attributes: Integer32, Property Must Be Supported

The minimum value acceptable for this property.

- **Property Maximum Integer32 {dmaProp_PropertyMaximumInteger32}**

Property Attributes: Integer32, Property Must Be Supported. The maximum value acceptable for this property.

4.1.35 dmaClass_PropertyDescriptionObject

A metadata class object describing an object property.

Class ID: dmaClass_PropertyDescriptionObject

Superclass: Property Description

4.1.35.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.35.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Property Description Object
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Cardinality	Yes	-	-	Yes	Integer32	Scalar	
Is Selectable	Yes	-	-	Yes	Boolean	Scalar	
Is Searchable	Yes	-	-	Yes	Boolean	Scalar	
Is Orderable	Yes	-	-	Yes	Boolean	Scalar	
Query Operator Descriptions	Yes	-	-	-	Object	List	Query Operator Description
Is System Generated	Yes	-	-	Yes	Boolean	Scalar	
Is Read Only	Yes	-	-	Yes	Boolean	Scalar	

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Is Value Required	Yes	-	-	Yes	Boolean	Scalar	
Is Hidden	Yes	-	-	Yes	Boolean	Scalar	
Property Default Object	Yes	-	-	-	Object	Scalar	
Property Selections Object	Yes	-	-	-	Object	List	
Required Class	Yes	-	-	-	Object	Scalar	Class Description
Reflective Property Id	Yes	-	-	-	ID	Scalar	

4.1.35.3 Description

The current object is an element in the Property List of a Class Description object. The current object describes a single property of the class being described by the Class Description object. The base datatype of the property being described is object.

4.1.35.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Property Description
- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Property Description
- **Ids {dmaProp_Ids}**
Property Inherited from Property Description

- **Data Type {dmaProp_DataType}**
Property Inherited from Property Description
- **Cardinality {dmaProp_Cardinality}**
Property Inherited from Property Description
- **Is Selectable {dmaProp_IsSelectable}**
Property Inherited from Property Description
- **Is Searchable {dmaProp_IsSearchable}**
Property Inherited from Property Description
- **Is Orderable {dmaProp_IsOrderable}**
Property Inherited from Property Description
- **Query Operator Descriptions {dmaProp_QueryOperatorDescriptions}**
Property Inherited from Property Description
- **Is System Generated {dmaProp_IsSystemGenerated}**
Property Inherited from Property Description
- **Is Read Only {dmaProp_IsReadOnly}**
Property Inherited from Property Description
- **Is Value Required {dmaProp_IsValueRequired}**
Property Inherited from Property Description
- **Is Hidden {dmaProp_IsHidden}**
Property Inherited from Property Description
- **Property Default Object {dmaProp_PropertyDefaultObject}**

Property Attributes: Object, Property Must Be Supported

The default object for this property.
- **Property Selections Object {dmaProp_PropertySelectionsObject}**

Property Attributes: List Of Object, Property Must Be Supported

The list of objects that may be assigned to this property.

If a selection list is provided, only an object existing in the list can be assigned to this property.
- **Required Class {dmaProp_RequiredClass}**

Property Attributes: Objects of Class 'Class Description', Property Must Be Supported

Specifies the class to which all objects delivered by the enumeration will conform.

- **Reflective Property Id {dmaProp_ReflectivePropertyId}**

Property Attributes: ID, Property Must Be Supported

The current Property Description Object describes an object valued property P of some class A. Suppose that objects of class A are independently persistent. Suppose also that objects of the required class of P, B, are also independently persistent. Then we say that obtaining the value of P is "navigating" from one object instance to another. In some such cases, a property in class B may provide a reverse navigation path, from the instance of class B to an instance of Class A having the same OIID as the original object. The property of B that provides this reverse path is termed the reflective property, and the Reflective Property Id identifies this property by means of one of its Ids. (Any of the Ids of the reflective property).

An invariant holds amongst the classes and properties involved in a reflective relationship. Let P be the described property (in class A) and Q be its reflective property (in class B, the required class of P). Further, let C be the class that introduces P (a superclass of A if P is inherited, and otherwise A itself). The required class of Q must be a subclass of C for which A is a subclass, and the reflective property of Q must be P.

4.1.36 dmaClass_PropertyDescriptionString

A metadata class object describing a string property.

Class ID: dmaClass_PropertyDescriptionString

Superclass: Property Description

4.1.36.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.36.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Property Description String
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Cardinality	Yes	-	-	Yes	Integer32	Scalar	
Is Selectable	Yes	-	-	Yes	Boolean	Scalar	
Is Searchable	Yes	-	-	Yes	Boolean	Scalar	
Is Orderable	Yes	-	-	Yes	Boolean	Scalar	
Query Operator Descriptions	Yes	-	-	-	Object	List	Query Operator Description
Is System Generated	Yes	-	-	Yes	Boolean	Scalar	
Is Read Only	Yes	-	-	Yes	Boolean	Scalar	

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
Is Value Required	Yes	-	-	Yes	Boolean	Scalar	
Is Hidden	Yes	-	-	Yes	Boolean	Scalar	
Property Default String	Yes	-	-	-	String	Scalar	
Property Selections String	Yes	-	-	-	String	List	
Maximum Length String	Yes	-	-	Yes	Integer32	Scalar	

4.1.36.3 Description

The current object is an element in the Property List of a Class Description object. The current object describes a single property of the class being described by the Class Description object. The base datatype of the property being described is string.

4.1.36.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Property Description
- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Property Description
- **Ids {dmaProp_Ids}**
Property Inherited from Property Description

- **Data Type {dmaProp_DataType}**
Property Inherited from Property Description
- **Cardinality {dmaProp_Cardinality}**
Property Inherited from Property Description
- **Is Selectable {dmaProp_IsSelectable}**
Property Inherited from Property Description
- **Is Searchable {dmaProp_IsSearchable}**
Property Inherited from Property Description
- **Is Orderable {dmaProp_IsOrderable}**
Property Inherited from Property Description
- **Query Operator Descriptions {dmaProp_QueryOperatorDescriptions}**
Property Inherited from Property Description
- **Is System Generated {dmaProp_IsSystemGenerated}**
Property Inherited from Property Description
- **Is Read Only {dmaProp_IsReadOnly}**
Property Inherited from Property Description
- **Is Value Required {dmaProp_IsValueRequired}**
Property Inherited from Property Description
- **Is Hidden {dmaProp_IsHidden}**
Property Inherited from Property Description
- **Property Default String {dmaProp_PropertyDefaultString}**

Property Attributes: String, Property Must Be Supported. The default string value for this property.
- **Property Selections String {dmaProp_PropertySelectionsString}**

Property Attributes: List Of String, Property Must Be Supported

The list of string values that may be assigned to this property.

If a selection list is provided, only a value existing in the list can be assigned to this property.
- **Maximum Length String {dmaProp_MaximumLengthString}**

Property Attributes: Integer32, Property Must Be Supported, Value Required

The maximum length, in characters, excluding the null terminator, of a value for the described property. The value -1 (DMA_NO_MAXIMUM) indicates no restriction on length.

4.1.37 dmaClass_Query

The complete definition of a query, including control settings.

Class ID: dmaClass_Query

Superclass: DMA

4.1.37.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.37.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Query Root	Yes	-	-	Yes	Object	Scalar	Query Root
Batch Size Hint	-	-	-	-	Integer32	Scalar	
Maximum Result Items	-	-	-	-	Integer32	Scalar	
Time Limit	-	-	-	-	Integer32	Scalar	
Collation Sequence Id	Yes	-	-	-	ID	Scalar	

4.1.37.3 Description

Query objects are passed to the ExecuteSearch method. The main part of a Query is the Query Root property, which contains the entire description of the query.

4.1.37.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Query Root {dmaProp_QueryRoot}**

Property Attributes: Objects of Class 'Query Root', Property Must Be Supported, Value Required

This specifies the query to be performed.

- **Batch Size Hint {dmaProp_BatchSizeHint}**

Property Attributes: Integer32

This integer property is relevant for the case that result rows are generated incrementally on demand in groups of N. The value of this property is a suggestion for N. If results are generated all at once, this property is simply ignored.

- **Maximum Result Items {dmaProp_MaximumResultItems}**

Property Attributes: Integer32

The value of this property indicates to the scope that it need not return more than the specified number of rows. If the result set is truncated as a result of reaching this upper limit, the result code DMARC_MAX_ROWS is returned. The value of this property is passed to all the component scopes

- **Time Limit {dmaProp_TimeLimit}**

Property Attributes: Integer32

This integer property gives a hint as to the maximum number of elapsed seconds between the time ExecuteSearch is called and the time query result items need no longer be returned. In the case of merged scopes, the

hint applies to each of the component scopes separately as well as to the merged scope itself.

When a result set is truncated because of this time limit being exceeded, `IdmaResultRow::GetNextResultRow` returns `DMARC_TIMEOUT`. If results are truncated for multiple different reasons (e.g., `DMARC_MAX_ROWS` and `DMARC_TIMEOUT`), the result `DMARC_TRUNCATED` is returned.

- **Collation Sequence Id {dmaProp_CollationSequenceld}**

Property Attributes: ID, Property Must Be Supported

This property specifies the text collation sequence to be used for all comparisons of string properties in all scopes executing the query.

4.1.38 dmaClass_QueryConstant

The purpose of this class is to factor out properties common to each query constant class, and to each query parameter class.

Class ID: dmaClass_QueryConstant

Superclass: Query Node

4.1.38.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.38.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	

4.1.38.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

Specifies the data type of the constant value. The permitted values are DMA_DATATYPE_BINARY, DMA_DATATYPE_BOOLEAN, DMA_DATATYPE_DATETIME, DMA_DATATYPE_FLOAT64, DMA_DATATYPE_ID, DMA_DATATYPE_INTEGER32 and DMA_DATATYPE_STRING.

Data Types are described at the end of this chapter.

- **Constant Is List {dmaProp_ConstantIsList}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

If set, this Boolean property indicates that the constant is a list of constants. If reset, the constant is a singleton.

4.1.39 dmaClass_QueryConstantBinaries

This class is for lists of binary constants in query expressions.

Class ID: dmaClass_QueryConstantBinaries

Superclass: Query Constant

4.1.39.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.39.2 Properties

Name	Impl. Req'd	Sys-tem Gen'd	Read-Only	Value Req'd	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip- tion
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Binaries
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Bina- ries	Yes	-	-	-	Binary	List	

4.1.39.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Binaries {dmaProp_ConstantValueBinaries}**

Property Attributes: List Of Binary, Property Must Be Supported

The list of constant Binary values. If the list is empty, a scope which receives the object should treat it as specifying the "unknown" Binary value of list cardinality.

4.1.40 dmaClass_QueryConstantBinary

This class is for singleton constants in query expressions.

Class ID: dmaClass_QueryConstantBinary

Superclass: Query Constant

4.1.40.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.40.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Binary
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Binary	Yes	-	-	-	Binary	Scalar	

4.1.40.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Binary {dmaProp_ConstantValueBinary}**

Property Attributes: Binary, Property Must Be Supported

The constant binary value. If no value is set for this property, a scope which receives the object should treat it as specifying the "unknown" binary value.

4.1.41 dmaClass_QueryConstantBoolean

This class is for Boolean constants in query expressions.

Class ID: dmaClass_QueryConstantBoolean

Superclass: Query Constant

4.1.41.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.41.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Boolean
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Boolean	Yes	-	-	-	Boolean	Scalar	

4.1.41.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Boolean {dmaProp_ConstantValueBoolean}**

Property Attributes: Boolean, Property Must Be Supported

The constant Boolean value. If no value is set for this property, a scope which receives the object should treat it as specifying the "unknown" Boolean value.

4.1.42 dmaClass_QueryConstantBooleans

This class is for lists of Boolean constants in query expressions.

Class ID: dmaClass_QueryConstantBooleans

Superclass: Query Constant

4.1.42.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.42.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Booleans
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Booleans	Yes	-	-	-	Boolean	List	

4.1.42.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Booleans {dmaProp_ConstantValueBooleans}**

Property Attributes: List Of Boolean, Property Must Be Supported

The list of constant Boolean values. If the list is empty, a scope which receives the object should treat it as specifying the "unknown" Boolean value of list cardinality.

4.1.43 dmaClass_QueryConstantDateTime

This class is for Date Time constants in query expressions.

Class ID: dmaClass_QueryConstantDateTime

Superclass: Query Constant

4.1.43.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.43.2 Properties

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip-tion
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Date Time
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer3 2	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Date Time	Yes	-	-	-	Date Time	Scalar	

4.1.43.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Date Time {dmaProp_ConstantValueDateTime}**

Property Attributes: Date Time, Property Must Be Supported

The constant Date Time value. If no value is set for this property, a scope which receives the object should treat it as specifying the "unknown" Date Time value.

4.1.44 dmaClass_QueryConstantDateTimes

This class is for lists of Date Time constants in query expressions.

Class ID: dmaClass_QueryConstantDateTimes

Superclass: Query Constant

4.1.44.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.44.2 Properties

Name	Impl. Req'ed	Sys-tem Gen'ed	Read-Only	Value Req'ed	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip-tion
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Date Times
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Date Times	Yes	-	-	-	Date Time	List	

4.1.44.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Date Times {dmaProp_ConstantValueDateTimes}**

Property Attributes: List Of Date Time, Property Must Be Supported

The list of constant Date Time values. If the list is empty, a scope which receives the object should treat it as specifying the "unknown" Date Time value of list cardinality.

4.1.45 dmaClass_QueryConstantFloat64

This class is for float64 constants in query expressions.

Class ID: dmaClass_QueryConstantFloat64

Superclass: Query Constant

4.1.45.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.45.2 Properties

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip-tion
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Float64
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Float64	Yes	-	-	-	Float64	Scalar	

4.1.45.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Float64 {dmaProp_ConstantValueFloat64}**

Property Attributes: Float64, Property Must Be Supported

The constant Float64 value. If no value is set for this property, a scope which receives the object should treat it as specifying the "unknown" Float64 value.

4.1.46 dmaClass_QueryConstantFloat64s

This class is for lists of float64 constants in query expressions.

Class ID: dmaClass_QueryConstantFloat64s

Superclass: Query Constant

4.1.46.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.46.2 Properties

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip- tion
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Float64s
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Float64s	Yes	-	-	-	Float64	List	

4.1.46.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Float64s {dmaProp_ConstantValueFloat64s}**

Property Attributes: List Of Float64, Property Must Be Supported

The list of constant Float64 values. If the list is empty, a scope which receives the object should treat it as specifying the "unknown" Float64 value of list cardinality.

4.1.47 dmaClass_QueryConstantId

This class is for DmaId constants in query expressions.

Class ID: dmaClass_QueryConstantId

Superclass: Query Constant

4.1.47.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.47.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Id
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Id	Yes	-	-	-	ID	Scalar	

4.1.47.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Id {dmaProp_ConstantValueId}**

Property Attributes: ID, Property Must Be Supported

The constant Id value. If no value is set for this property, a scope which receives the object should treat it as specifying the "unknown" Id value.

4.1.48 dmaClass_QueryConstantIds

This class is for lists of DmalDs in query expressions.

Class ID: dmaClass_QueryConstantIds

Superclass: Query Constant

4.1.48.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.48.2 Properties

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip-tion
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Ids
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Ids	Yes	-	-	-	ID	List	

4.1.48.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Ids {dmaProp_ConstantValueIds}**

Property Attributes: List Of ID, Property Must Be Supported

The list of constant Ids values. If the list is empty, a scope which receives the object should treat it as specifying the "unknown" Ids value of list cardinality.

4.1.49 dmaClass_QueryConstantInteger32

This class is for integer32 constants in query expressions.

Class ID: dmaClass_QueryConstantInteger32

Superclass: Query Constant

4.1.49.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.49.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Integer32
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Integer32	Yes	-	-	-	Integer32	Scalar	

4.1.49.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Integer32 {dmaProp_ConstantValueInteger32}**

Property Attributes: Integer32, Property Must Be Supported

The constant Integer32 value. If no value is set for this property, a scope which receives the object should treat it as specifying the "unknown" Integer32 value.

4.1.50 dmaClass_QueryConstantInteger32s

This class is for lists of integer32 constants in query expressions.

Class ID: dmaClass_QueryConstantInteger32s

Superclass: Query Constant

4.1.50.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.50.2 Properties

Name	Impl. Req'd	Sys-tem Gen'd	Read-Only	Value Req'd	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Integer32s
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Integer32s	Yes	-	-	-	Integer32	List	

4.1.50.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Integer32s {dmaProp_ConstantValueInteger32s}**

Property Attributes: List Of Integer32, Property Must Be Supported

The list of constant Integer32 values. If the list is empty, a scope which receives the object should treat it as specifying the "unknown" Integer32 value of list cardinality.

4.1.51 dmaClass_QueryConstantString

This class is for singleton constants in query expressions.

Class ID: dmaClass_QueryConstantString

Superclass: Query Constant

4.1.51.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.51.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant String
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value String	Yes	-	-	-	String	Scalar	

4.1.51.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value String {dmaProp_ConstantValueString}**

Property Attributes: String, Property Must Be Supported

The constant String value. If no value is set for this property, a scope which receives the object should treat it as specifying the "unknown" String value.

4.1.52 dmaClass_QueryConstantStrings

This class is for a list of string constants in a query expression.

Class ID: dmaClass_QueryConstantStrings

Superclass: Query Constant

4.1.52.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.52.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Constant Strings
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Constant Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Constant Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Constant Value Strings	Yes	-	-	-	String	List	

4.1.52.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Constant Data Type {dmaProp_ConstantDataType}**
Property Inherited from Query Constant
- **Constant Is List {dmaProp_ConstantIsList}**
Property Inherited from Query Constant
- **Constant Value Strings {dmaProp_ConstantValueStrings}**

Property Attributes: List Of String, Property Must Be Supported

The list of constant String values. If the list is empty, a scope which receives the object should treat it as specifying the "unknown" String value of list cardinality.

4.1.53 dmaClass_QueryJoinOperator

This object class represents a join operation.

Class ID: dmaClass_QueryJoinOperator

Superclass: Query Nonterminal Node

4.1.53.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.53.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Join Operator
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Operands	Yes	-	-	Yes	Object	List	Query Node
Join Operator Id	Yes	-	-	Yes	ID	Scalar	

4.1.53.3 Description

A Query Join Operator object is used to define a join operation as a component of the From Expression of a query.

Operand 0 is the first searchable class instance to be joined, or another object instance of this class. Operand 1 is the second searchable class to be joined. Operand 2, if applicable, is the "on" condition of the join. This must be a Query Operator node specifying an operator yielding a singleton result of type Boolean and having a join participation level of DMA_JOIN_PARTICIPATION_OPERAND.

4.1.53.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Operands {dmaProp_Operands}**
Property Inherited from Query Nonterminal Node
- **Join Operator Id {dmaProp_JoinOperatorId}**

Property Attributes: ID, Property Must Be Supported, Value Required

Specifies the id of the join operator to be applied to the operands given by dmaProp_Operands. The Query Operator Description object for the operator having that id must have the value DMA_JOIN_PARTICIPATION_OPERATOR as the value for its Join Participation property. The following are Ids in DMA header files representing the possible join operator Id's.

Join Operators are described at the end of this chapter.

4.1.54 dmaClass_QueryNode

The base class for most Query-related objects.

Class ID: dmaClass_QueryNode

Superclass: DMA

4.1.54.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]

4.1.54.2 Properties

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip-tion
This	-	Yes	Yes	Yes	Object	Scalar	Query Node
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	

4.1.54.3 Description

Query Node objects are non-persistent helper objects used to form From Expressions, Selections, Orderings, and Query Expression. They are basically just a collection of properties. Object valued properties of Query Node objects are maintained by reference. This means that an internal interface pointer is kept internally in Query Node objects. Query Node objects form a hierarchy rooted at the Query object. The hierarchy can be traversed from the root downward by getting the value of object valued properties.

4.1.54.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA

- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA

4.1.55 dmaClass_QueryNonterminalNode

This class exists to provide a place to factor out the dmaProp_Operands property of query node objects.

Class ID: dmaClass_QueryNonterminalNode

Superclass: Query Node

4.1.55.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.55.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Nonterminal Node
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Operands	Yes	-	-	-	Object	List	Query Node

4.1.55.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA

- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Operands {dmaProp_Operands}**

Property Attributes: List Of Objects of Class 'Query Node', Property Must Be Supported

This list valued property refers to the operands of a query node object. The list elements are interface pointers. Thus, the operand objects are referred to by reference. Thus, if you ask for operand zero, you will get the same one every time.

4.1.56 dmaClass_QueryOperandDescription

The Query Operand Description class is the metadata class that defines the signatures of operands used in a query expression. Note that this class does not inherit from the Metadata class, even though it is metadata.

Class ID: dmaClass_QueryOperandDescription
Superclass: DMA

4.1.56.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.56.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Operand Description
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Operand Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Allows Singleton	Yes	Yes	Yes	Yes	Boolean	Scalar	
Allows List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Allows Constant	Yes	Yes	Yes	Yes	Boolean	Scalar	
Allows Property	Yes	Yes	Yes	Yes	Boolean	Scalar	
Allows Expression	Yes	Yes	Yes	Yes	Boolean	Scalar	

4.1.56.3 Description

This class describes operands of operators. The data type of the operand is well known and is shown in the well known operator description, but the Boolean flag values are dependent on what the particular document space supports.

4.1.56.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Operand Data Type {dmaProp_OperandDataType}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

The value of this property is the datatype of the operand.

The legal values for the datatypes of the operands of operators, and for the result of operators in query expressions are as follows.

Data Types are described at the end of this chapter.

- **Allows Singleton {dmaProp_AllowsSingleton}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

This Boolean property indicates whether the operand can be a singleton.

- **Allows List {dmaProp_AllowsList}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

This Boolean property indicates whether the operand can be a list.

At least one of dmaProp_AllowsSingleton and dmaProp_AllowsList must be TRUE.

- **Allows Constant {dmaProp_AllowsConstant}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

This Boolean property indicates whether the operand can be a constant.

- **Allows Property {dmaProp_AllowsProperty}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

This Boolean property indicates whether the operand can be a property of a persistent object.

- **Allows Expression {dmaProp_AllowsExpression}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

This Boolean property indicates whether the operand can be the value of an expression.

At least one of dmaProp_AllowsConstant, dmaProp_AllowsProperty and dmaProp_AllowsExpression must be TRUE.

4.1.57 dmaClass_QueryOperator

This class represents operators in a query expression.

Class ID: dmaClass_QueryOperator

Superclass: Query Nonterminal Node

4.1.57.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.57.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Operator
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Operands	Yes	-	-	-	Object	List	Query Node
Query Operator Id	Yes	-	-	Yes	ID	Scalar	

4.1.57.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA

- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Operands {dmaProp_Operands}**
Property Inherited from Query Nonterminal Node
- **Query Operator Id {dmaProp_QueryOperatorId}**

Property Attributes: ID, Property Must Be Supported, Value Required

Specifies the id of the operator to be applied to the operands given by dmaProp_Operands.

The Query Operator Description object for the operator having that id must have the value DMA_JOIN_PARTICIPATION_NONE or DMA_JOIN_PARTICIPATION_OPERAND as the value for its Join Participation property.

The Ids of query operators are well known DMA-defined Ids.

Query Operators are described at the end of this chapter.

4.1.58 dmaClass_QueryOperatorDescription

The Query Operator Description class is the metadata class that defines the signatures of operators used in a Query Expression.

Class ID: dmaClass_QueryOperatorDescription

Superclass: Metadata

4.1.58.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.58.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Operator Description
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Display Name	Yes	-	-	Yes	String	Scalar	
Descriptive Text	Yes	-	-	Yes	String	Scalar	
Ids	Yes	-	-	Yes	ID	List	
Result Type	Yes	Yes	Yes	Yes	Integer32	Scalar	
Is List	Yes	Yes	Yes	Yes	Boolean	Scalar	
Minimum Operands	Yes	Yes	Yes	Yes	Integer32	Scalar	
Maximum Operands	Yes	Yes	Yes	Yes	Integer32	Scalar	
Is Safe to Eliminate	Yes	Yes	Yes	Yes	Boolean	Scalar	
Operand Descriptions	Yes	Yes	Yes	-	Object	List	Query Operand Description

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Join Participation	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.58.3 Description

Since operators are identified by Dmalds, new query operators can be introduced without any collisions on operator ID's. The operator Id is found in dmaProp_Ids.

4.1.58.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Display Name {dmaProp_DisplayName}**
Property Inherited from Metadata
- **Descriptive Text {dmaProp_DescriptiveText}**
Property Inherited from Metadata
- **Ids {dmaProp_Ids}**
Property Inherited from Metadata
- **Result Type {dmaProp_ResultType}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

Specifies the data type of the constant value. The permitted values are DMA_DATATYPE_BINARY, DMA_DATATYPE_BOOLEAN, DMA_DATATYPE_DATETIME, DMA_DATATYPE_FLOAT64, DMA_DATATYPE_

ID, DMA_DATATYPE_INTEGER32, DMA_DATATYPE_OBJECT, DMA_DATATYPE_STRING and DMA_DATATYPE_CLASS.

Data Types are described at the end of this chapter.

- **Is List {dmaProp_IsList}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

If TRUE, the output result of the operator is a list. If FALSE, the output result of the operator is a scalar.

- **Minimum Operands {dmaProp_MinimumOperands}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

This is the minimum number of operands required for the operator. The value must be non-negative.

- **Maximum Operands {dmaProp_MaximumOperands}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

This is the maximum number of operands allowed for the operator.

If Minimum Operands and Maximum Operands are both zero, then the operator takes no operands. An example of an operator with no operands would be TODAY in SQL: The value is today's date and time at the instant the query is executed.

If the value of Maximum Operands is -1 (DMA_NO_MAXIMUM), the operator can take an unlimited number of operands.

- **Is Safe to Eliminate {dmaProp_IsSafeToEliminate}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

This boolean property indicates to a merged scope whether this operator and its subtree of operands can safely be pruned from a query parse tree delivered to a component scope using the rules of three valued logic, in the case that one or more of its operands is undefined relative to that component scope.

An operator is safe to eliminate (implying a value of TRUE for this property) if it always yields the "unknown" result if any of its operands is undefined. This allows the merged scope to replace the operator node and its subtree of operands the "unknown" value of the appropriate datatype. An operator

is not safe to eliminate (implying a value of FALSE for this property) if its outcome depends on the values of the other (defined) operands. In this case, the operator node and subtree cannot usually be eliminated by the merged scope; it must be delivered to the component scope in order for the expression to be properly evaluated. Most relational operators (e.g.: EqualString) fall into the former category, while the logical operators And and Or fall into the latter.

Note that the issue of elimination of operators can only arise when a merged scope is created using union rules.

DMA defines a number of "well known" query operators and specifies the required setting for this property for them. A conformant DMA scope implementation must deliver the specified value for any of the well known operators that it implements.

If a scope supports a non-eliminable operator, the operator must allow any operand to be a constant query node for which the value of the property is not set. It must treat such a constant operand node as supplying the "undefined" value for the constant.

- **Operand Descriptions {dmaProp_OperandDescriptions}**

Property Attributes: List Of Objects of Class 'Query Operand Description', Property Must Be Supported, System Generated, Read-Only

A list of Query Operand Description objects describing the operands of the operator. If the operator has no operands, the list will be empty. Otherwise, element 0 of this list describes the first operand of the operator. Element 1 describes the second operand of the operator, etc. The last element in the list describes the operand at that list index and all subsequent operands, if any.

For example, the "And" and "AddInteger32" operators are associative and commutative, so they are N-ary operators, $N \geq 2$. Thus, Minimum Operands is 2, and Maximum Operands is -1, and there is only one element in the Operand Descriptions list, since all the operands have the same description.

- **Join Participation {dmaProp_JoinParticipation}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

Specifies the extent to which this operator can participate in a join expression (in the From clause of a query) and determines how to construct instances of the operator.

The following are the permitted values for this property:

0 - DMA_JOIN_PARTICIPATION_NONE - The operator, which is a normal operator, cannot appear in a From expression. Instances of this operator are of class dmaClass_QueryOperator.

1 - DMA_JOIN_PARTICIPATION_OPERAND - The operator, which is a normal operator, can appear as the third operand (the "on" condition) of a join operator. Instances of this operator are of class dmaClass_QueryOperator. For an operator of this type the Result Type property of the QueryOperatorDescription object must specify Boolean and the Is List property must be FALSE.

2 - DMA_JOIN_PARTICIPATION_OPERATOR - The operator is a join operator which joins classes within a single document space scope. (All of the well known join operators in DMA 1.0 are of this type). It may appear as the root of a From expression or as the first operand argument of another join operator. Instances of this operator are of class dmaClass_QueryJoinOperator. For an operator of this type the Result Type property of the QueryOperatorDescription object must specify Class and the Is List property must be FALSE

4.1.59 dmaClass_QueryOrderByNode

The elements in an Order By list must be of this class. Document spaces do not have to support this class. And if it is supported, the Order By list is optional for any given query.

Class ID: dmaClass_QueryOrderByNode

Superclass: Query Node

4.1.59.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.59.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Order By Node
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Selections Index	Yes	-	-	Yes	Integer32	Scalar	
Descending Requested	-	-	-	-	Boolean	Scalar	

4.1.59.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Selections Index {dmaProp_SelectionIndex}**

Property Attributes: Integer32, Property Must Be Supported, Value Required

The index, in the Selections list of the Query Root, of the property to order by.

- **Descending Requested {dmaProp_DescendingRequested}**

Property Attributes: Boolean

If the value of this property is DMA_TRUE, the order is descending. Otherwise, the order is ascending.

4.1.60 dmaClass_QueryProperty

The purpose of this class is to designate a property of a persistent object involved in a query. Thus, leaf nodes of the Query Expression parse tree that reference properties are of this class.

Class ID: dmaClass_QueryProperty

Superclass: Query Node

4.1.60.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.60.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Property
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Searchable Class Occurrence	Yes	-	-	Yes	Integer32	Scalar	
Property Id	Yes	-	-	Yes	ID	Scalar	

4.1.60.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Searchable Class Occurrence {dmaProp_SearchableClassOccurrence}**

Property Attributes: Integer32, Property Must Be Supported, Value Required

This is the occurrence number of the searchable class in the current query.

In each query, the searchable class occurrences are assigned non negative integers by the client. These integers are unique in the context of the current query. This is necessary because the same searchable class can occur more than once in the From Expression's of the query. References to properties in the query are tied to a particular searchable class occurrence.

- **Property Id {dmaProp_PropertyId}**

Property Attributes: ID, Property Must Be Supported, Value Required

Specifies the property of the searchable class. The value may be any one of the identifiers for the desired property.

4.1.61 dmaClass_QueryResultRow

This object is a result row from the result set object of a query. Thus, this is the end product of a query.

Class ID: dmaClass_QueryResultRow

Superclass: DMA

4.1.61.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties

4.1.61.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Result Row
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Select List Offset	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.61.3 Description

Note that result row objects just hold properties: they do not support the Idma-Connection interface, so they are never a scratchpad object for an independently persistable object.

4.1.61.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Select List Offset {dmaProp_SelectListOffset}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

The value of this property gives an index value suitable for passing to a GetPropVal*ByIndex method. The property value at that index will be the value of the first property in the Selections list of the query which generated this result row. The remaining Selections properties appear in order with consecutively higher indices from this index.

The value of dmaProp_SelectListOffset will be the same for all result rows belonging to a single result set, but may differ from result set to result set.

4.1.62 dmaClass_QueryResultSet

Objects of this class are the root of a tree of objects stitched together by reference. The root object of the main query and the root object of all subqueries (if any) are of this class.

Class ID: dmaClass_QueryResultSet

Superclass: Enumeration

4.1.62.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaResultSet

4.1.62.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Result Set
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Data Type	Yes	Yes	Yes	Yes	Integer32	Scalar	

4.1.62.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Data Type {dmaProp_DataType}**
Property Inherited from Enumeration

4.1.63 dmaClass_QueryRoot

The root of a tree of Query Node objects that represents the main query or a subquery of a query expression.

Class ID: dmaClass_QueryRoot

Superclass: Query Node

4.1.63.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.63.2 Properties

Name	Impl. Req'd	Sys-tem Gen'e d	Read-Only	Value Req'e d	Type	Cardinal-ity	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Descrip- tion
This	-	Yes	Yes	Yes	Object	Scalar	Query Root
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
From Expression	Yes	-	-	Yes	Object	Scalar	Query Node
Selections	Yes	-	-	-	Object	List	Query Property
Query Expression	Yes	-	-	-	Object	Scalar	Query Node
Orderings	Yes	-	-	-	Object	List	Query Order By Node
Distinct Rows Re- quested	-	-	-	-	Boolean	Scalar	

4.1.63.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA

- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **From Expression {dmaProp_FromExpression}**

Property Attributes: Objects of Class 'Query Node', Property Must Be Supported, Value Required

The searchable classes that are joined together are listed in the From Expression object, along with the type of join, and the columns being joined.

The value for this property must be of class dmaClass_SearchableClass or dmaClass_QueryJoinOperator.

- **Selections {dmaProp_Selections}**

Property Attributes: List Of Objects of Class 'Query Property', Property Must Be Supported

This object valued property is a list of objects of class dmaClass_QueryNode. On 1.0, all the list elements must be of class dmaClass_QueryProperty.

The select list may not be null for the main query. It must be null for subqueries under the EXISTS operator. The select list must be non-null for subqueries under the IN operators, and, it must contain exactly one element of class dmaClass_QueryProperty, which is a base datatype property of a persistent object.

Data Types are described at the end of this chapter.

- **Query Expression {dmaProp_QueryExpression}**

Property Attributes: Objects of Class 'Query Node', Property Must Be Supported

The value of this property is the query root of the main query expression. This is analogous to the "where" clause in a SQL "select" statement.

If the query expression is NULL, no filtering is done, and all persistent objects as determined by the From Expression are returned as result rows.

- **Orderings {dmaProp_Orderings}**

Property Attributes: List Of Objects of Class 'Query Order By Node', Property Must Be Supported

This property is a list of objects of class dmaClass_QueryOrderByNode. On 1.0, for each list element, the property must occur in the dmaProp_Selections list. Each list element specifies a property to order the result rows on, and whether the order is ascending or descending. The ordering is first on list element 0, then within that on list element 1, etc.

- **Distinct Rows Requested {dmaProp_DistinctRowsRequested}**

Property Attributes: Boolean

If this Boolean property is true, then all result rows returned have a unique set of values for the properties designated in the Property Selections list.

In other words, duplicate result rows are suppressed, and only properties in the Property Selections list are considered in the comparison. Equality is determined using the well known "equal" operator defined by DMA that is appropriate to the datatype and cardinality of the Property Selections list elements. Note that enumeration objects are always considered distinct. Two null values are considered as equal for the purpose of determining distinctness.

4.1.64 dmaClass_QuerySearchableClass

Objects of this class designate a searchable class in a Document space. Objects of this class appear in the From Expression of queries and subqueries.

Class ID: dmaClass_QuerySearchableClass

Superclass: Query Node

4.1.64.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties

4.1.64.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Query Searchable Class
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Searchable Class Id	Yes	-	-	Yes	ID	Scalar	
Searchable Class Occurrence	Yes	-	-	Yes	Integer32	Scalar	
Include Subclasses Requested	-	-	-	Yes	Boolean	Scalar	

4.1.64.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA

- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Searchable Class Id {dmaProp_SearchableClassId}**

Property Attributes: ID, Property Must Be Supported, Value Required

Specifies the searchable class. The value may be any one of the identifiers for the desired class.

- **Searchable Class Occurrence {dmaProp_SearchableClassOccurrence}**

Property Attributes: Integer32, Property Must Be Supported, Value Required

This is the occurrence number of the searchable class in the current query.

In each query, the searchable class occurrences are assigned non negative integers by the client. These integers are unique in the context of the current query. This is necessary because the same searchable class can occur more than once in the From Expression's of the query. References to properties in the query are tied to a particular searchable class occurrence.

- **Include Subclasses Requested {dmaProp_IncludeSubclassesRequested}**

Property Attributes: Boolean, Value Required

This property controls whether a search includes subclasses of the class specified.

If the value of this property is not set or is set to FALSE, then the only class searched is the class whose Id is the value of dmaProp_SearchableClassId.

If the value of this property is TRUE, then the class whose Id is the value of dmaProp_SearchableClassId and all subclasses of that class are searched, as long as the document space supports this feature. The dmaProp_HasIncludeSubclasses property of the Class Description object indicates whether this subclass search feature is supported by a particular

class. Note that if dmaProp_SearchableClassId is equal to dmaClass_DocVersion, and the value of this property and dmaProp_HasIncludeSubclasses of the Class Description are TRUE, then all document classes are searched.

See the dmaQueryOperator_IsClass operator. This operator can be used to restrict the classes searched to a subset of the tree of classes rooted at the class indicated by the value of dmaProp_SearchableClassId.

4.1.65 dmaClass_ReferentialContainmentRelationship

Defines a referential containment relationship.

Class ID: dmaClass_ReferentialContainmentRelationship

Superclass: Containment Relationship

4.1.65.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]
- IdmaRelationshipOrdering [Optional]

4.1.65.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Referential Containment Relationship
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Head	Yes	-	-	Yes	Object	Scalar	Containable
Tail	Yes	-	-	Yes	Object	Scalar	Container

4.1.65.3 Description

This class models referential containment within a DMA DocSpace. It is a subclass of dmaClass_ContainmentRelationship and introduces no new properties. This class and dmsClass_DirectContainmentRelationship are needed in the model to allow them to have different metadata, in particular searchability.

4.1.65.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Head {dmaProp_Head}**
Property Inherited from Relationship
- **Tail {dmaProp_Tail}**
Property Inherited from Relationship

4.1.66 dmaClass_Relationship

This is the base class for binary relationships between independently persistable objects of a Document Space.

Class ID: dmaClass_Relationship

Superclass: DMA

4.1.66.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]
- IdmaRelationshipOrdering [Optional]

4.1.66.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Relationship
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Head	Yes	-	-	Yes	Object	Scalar	DMA
Tail	Yes	-	-	Yes	Object	Scalar	DMA

4.1.66.3 Description

The base class dmaClass_Relationship models an association between a pair of objects within a DMA DocSpace. This class is introduced to permit "edge data" related to the association to be maintained. The first type of association enabled by subclasses of dmaClass_Relationship is containment. Other relationship types are possible.

4.1.66.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Head {dmaProp_Head}**

Property Attributes: Objects of Class 'DMA', Property Must Be Supported, Value Required

The value of this property is the independently persistable object at the "Head" end of the binary relationship.

This is a property of a Relationship object which identifies the object logically at the head end of an arrow between two related objects. In the case of containment, it is the contained object. The Head property is paired reflectively with the Containers, or Parent property of that object, for referential or direct containment, respectively.

- **Tail {dmaProp_Tail}**

Property Attributes: Objects of Class 'DMA', Property Must Be Supported, Value Required

The object in a Relationship which is logically at the tail end of an arrow between two related objects.

In the case of containment, it is the containing object. The Tail property is paired reflectively with the Containees, or Children, property of that object, for referential or direct containment, respectively.

4.1.67 dmaClass_Rendition

Represents the content of the DocVersion in a particular format.

Class ID: dmaClass_Rendition

Superclass: DMA

4.1.67.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]

4.1.67.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Rendition
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Rendition Type	Yes	-	-	Yes	String	Scalar	
Content Elements Present	Yes	Yes	Yes	-	String	List	
Content Elements	Yes	-	-	-	Object	List	Content Element

4.1.67.3 Description

A Rendition contains the content of the Document Version in a particular file format. It manages and provides access to all of the content material for that rendition. The Rendition is intended to be sub-classed to support the specific characteristics of the various rendition types that exist (e.g., ASCII text, Word, PostScript, scanned image documents, SGML, OpenDoc, etc.).

A rendition object has a string valued property (rendition type) that differentiates the renditions of the document and is propagated by the system to the Renditions Present property of the parent document version.

Renditions are associated with a flat list of component content elements. The Content Elements Present property is a system-generated list of strings which represent the types of these components.

4.1.67.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Rendition Type {dmaProp_RenditionType}**

Property Attributes: String, Property Must Be Supported, Value Required

Type of rendition. A string indicating type of rendition. These strings have the syntax "rendition_type_space::typename". The syntax and semantics of the typename is specific to the rendition type space. DMA 1.0 defines one rendition type space, "MIME", whose values are legal MIME types as defined by the IANA. Examples are "MIME::application/postscript" or "MIME::text/plain". Other rendition type spaces are anticipated.

- **Content Elements Present {dmaProp_ContentElementsPresent}**

Property Attributes: List Of String, Property Must Be Supported, System Generated, Read-Only

List of the component types of the content elements contained within this rendition at the time it was last made persistent.

A system-generated (and read-only) ordered list of Strings which reflect the component types of the content elements in the rendition.

This property is static except over a Refresh of the parent DocVersion and does not dynamically track the state of the Content Elements list. Therefore, the client application may not in general assume that the number or order of elements in this list correspond to that of the Content Elements list.

However, if neither the Content Elements list nor any of the content element objects it contains have been modified since the DocVersion was connected or last refreshed, then it is required that the two lists be synchronized.

- **Content Elements {dmaProp_ContentElements}**

Property Attributes: List Of Objects of Class 'Content Element', Property Must Be Supported

List of Content Elements contained within this Rendition

4.1.68 dmaClass_Reservation

Records the intention to perform a checkin of the versionable object to the specified version series.

Class ID: dmaClass_Reservation

Superclass: DMA

4.1.68.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]

4.1.68.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Reservation
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Reserved For	Yes	Yes	Yes	Yes	Object	Scalar	Version Series
New Version	-	Yes	Yes	-	Object	Scalar	Versionable

4.1.68.3 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA

- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Reserved For {dmaProp_ReservedFor}**

Property Attributes: Objects of Class 'Version Series', Property Must Be Supported, Value Required, System Generated, Read-Only

Indicates what version series the reservation object is associated with (before check-in)

This property specifies the Version Series for which this object is assigned as a Reservation. That Version Series object will have this object as its Reservation property value.

This Reservation object was created as the result of a SetCheckOutNext or SetReserveNext on the Version Series where the check-in will occur.

- **New Version {dmaProp_NewVersion}**

Property Attributes: Objects of Class 'Versionable', System Generated, Read-Only

For SetCheckOutNext, points to the "cloned" versionable object, for SetReserveNext this property is null.

If the SetCheckOutNext operation is not supported on any Version Series subclass, this property should be omitted from the runtime metadata.

4.1.69 dmaClass_Scope

A Scope object provides access to the query capabilities of a document space or collection of document spaces.

Class ID: dmaClass_Scope

Superclass: DMA

4.1.69.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaObjectFactory
- IdmaScope

4.1.69.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Scope
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Searchable Class Descriptions	Yes	Yes	Yes	Yes	Object	List	Class Description
Query Construction Class Descriptions	Yes	Yes	Yes	Yes	Object	List	Class Description
Operators	Yes	Yes	Yes	-	Object	List	Query Operator Description
Has Arbitrary Order By	Yes	Yes	Yes	Yes	Boolean	Scalar	
Has Distinct	Yes	Yes	Yes	Yes	Boolean	Scalar	
Component Scopes	-	Yes	Yes	-	Object	List	Scope
Collation Sequence Ids	Yes	Yes	Yes	-	ID	List	
Has Elimination	Yes	Yes	Yes	Yes	Boolean	Scalar	

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Metadata Epoch	-	Yes	Yes	-	ID	Scalar	

4.1.69.3 Description

The scope of a query may be a single Document Space, or span multiple Document Spaces. DMA distributes the query across the scope, resolves any differences among Document Spaces, and integrates the result set.

4.1.69.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Searchable Class Descriptions {dmaProp_SearchableClassDescriptions}**

Property Attributes: List Of Objects of Class 'Class Description', Property Must Be Supported, Value Required, System Generated, Read-Only

The value of this property is a list of Class Description objects, one for each class that can be referenced in queries using the current Scope.

- **Query Construction Class Descriptions {dmaProp_QueryConstructionClassDescriptions}**

Property Attributes: List Of Objects of Class 'Class Description', Property Must Be Supported, Value Required, System Generated, Read-Only

This is a list of class descriptions for the infrastructure classes that can be used to construct a valid query parse tree for this scope.

When constructing a parse tree object for a query that will be submitted to the `IdmaScope::ExecuteSearch` method of the current scope object, the `IdmaClassDescription::CreateInstance()` method of the appropriate Class Description object in this list can always be used to create the object. It is a requirement on all Scope objects that the list of Class Description objects in this list include every class needed to build a valid query parse tree, and that the `IdmaClassDescription::CreateInstance()` method will succeed, resources permitting. For example, the root object of the parse tree is of class `dmaClass_Query`. There is always a class description for `dmaClass_Query` in this list, and the `CreateInstance()` method of that class description must succeed in creating an instance of the class.

- **Operators {dmaProp_Operators}**

Property Attributes: List Of Objects of Class 'Query Operator Description', Property Must Be Supported, System Generated, Read-Only

The value of this property is a list of the Query Operator Description objects for every query operator supported by the current Scope object.

- **Has Arbitrary Order By {dmaProp_HasArbitraryOrderBy}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

This Boolean flag indicates whether this scope supports arbitrary OrderBy lists. This property should have the value `DMA_TRUE` if and only if all selectable properties in the scope are orderable, and the maximum permitted number of OrderBy list elements is greater than or equal to the maximum permitted number of Selections list elements.

This property gives information to the middleware software that implements merged scopes which can be useful if distinct query result rows are required. If the value of this property is `TRUE` for all component scopes of a merged scope, the middleware software is allowed to augment the Orderings list such that it specifies each property in the query's Selections list. This enables a merged scope to perform a merge instead of a sort of the result rows returned from its component scopes prior to returning its own result rows.

- **Has Distinct {dmaProp_HasDistinct}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

This Boolean flag indicates whether this scope supports `dmaProp_DistinctRowsRequested` on the QueryRoot object. If that flag is set, that indicates that duplicate result rows are to be suppressed. Normally, a sort of the result rows is required to accomplish duplicate result row suppression, and,

if any string properties are involved, text collation sequences come into play.

- **Component Scopes {dmaProp_ComponentScopes}**

Property Attributes: List Of Objects of Class 'Scope', System Generated, Read-Only

The list of Scope Objects from which the current Scope object was created.

A list of scope objects whose metadata was combined to create the current scope. If the Scope belongs to a Doc Space, then this property need not be supported, but if it is, the list will have zero elements.

- **Collation Sequence Ids {dmaProp_CollationSequenceIds}**

Property Attributes: List Of ID, Property Must Be Supported, System Generated, Read-Only

This property is the list of collation sequence Id's supported by the scope. The first Id in the list, if any, is the default. This list may be empty if and only if there are no orderable string properties on any searchable class in the Scope.

DMA defines a number of well known [collation sequences](#).

- **Has Elimination {dmaProp_HasElimination}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

If the value of this property is DMA_TRUE it indicates that, when requested to do so, the Scope is willing to apply the rules of three-valued logic to eliminate undefined elements from queries which are delivered to it, prior to executing the query. (An undefined element is a class, property or operator not present in the Scope's metadata). If the value of this property is DMA_FALSE, the Scope is not willing to perform such elimination, and will always generate an error if presented with a query containing undefined elements.

- **Metadata Epoch {dmaProp_MetadataEpoch}**

Property Attributes: ID, System Generated, Read-Only

The Metadata Epoch is a unique identifier for a particular state of the metadata space of the object.

Two instances may deliver the same value for the Metadata Epoch only if their metadata spaces are identical.

4.1.70 dmaClass_System

A DMA system is a collection of Document Spaces.

Class ID: dmaClass_System

Superclass: DMA

4.1.70.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaObjectFactory [Optional]
- IdmaOIID
- IdmaAuthentication [Optional]
- IdmaScopeFactory [Optional]
- IdmaServiceRegistry [Optional]
- IdmaSystem

4.1.70.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	System
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Character Set Encoding	Yes	Yes	Yes	Yes	Integer32	Scalar	
Locale Name	Yes	Yes	Yes	Yes	String	Scalar	
System OIID Prefix	Yes	Yes	Yes	Yes	String	Scalar	
Locale Names	Yes	Yes	Yes	Yes	String	List	
Display Name	Yes	-	-	Yes	String	Scalar	

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Descriptive Text	Yes	-	-	-	String	Scalar	
System Id	Yes	Yes	Yes	Yes	ID	Scalar	

4.1.70.3 Description

A DMA system may be comprised of a single Document Space or multiple Document Spaces, and may be supplied/implemented by different vendors.

Object valued properties cannot be accessed if the System object is in an unauthenticated state. (Lists that have non object-valued elements are not considered object-valued properties for this purpose, even though a DMA object interface is employed for delivery of those values.) Attempts to access an object valued property in an unauthenticated state will yield a DMARC_NOT_AUTHENTICATED return code.

A DMA system object is a source of system-specific, system-level services as well. These services are available via interfaces, such as IdmaScopeFactory and IdmaObjectFactory, which may be optionally available. Any locale-dependent interfaces that are provided for system-level services always honor the System object dmaProp_Locale property value established when the particular DMA System object is instantiated.

4.1.70.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Character Set Encoding {dmaProp_CharacterSetEncoding}**

Property Attributes: Integer32, Property Must Be Supported, Value Required, System Generated, Read-Only

Indicates the Character Set that this system uses.

- **Locale Name {dmaProp_LocaleName}**

Property Attributes: String, Property Must Be Supported, Value Required, System Generated, Read-Only

Indicates the name of the Locale this object is currently operating in.

- **System OIID Prefix {dmaProp_SystemOIIDPrefix}**

Property Attributes: String, Property Must Be Supported, Value Required, System Generated, Read-Only. The system prefix part to be used when constructing an OIID.

- **Locale Names {dmaProp_LocaleNames}**

Property Attributes: List Of String, Property Must Be Supported, Value Required, System Generated, Read-Only

The list of locales this object supports.

- **Display Name {dmaProp_DisplayName}**

Property Attributes: String, Property Must Be Supported, Value Required. A string containing a user-intelligible name for the class/property/... being described.

- **Descriptive Text {dmaProp_DescriptiveText}**

Property Attributes: String, Property Must Be Supported. Text documenting this object.

- **System Id {dmaProp_SystemId}**

Property Attributes: ID, Property Must Be Supported, Value Required, System Generated, Read-Only. The persistent Id that identifies this System.

4.1.71 dmaClass_Versionable

Base class for versionable classes

Class ID: dmaClass_Versionable

Superclass: Containable

4.1.71.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]
- IdmaVersionable [Optional]

4.1.71.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Versionable
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Parent	-	Yes	Yes	-	Object	Scalar	Direct Containment Relationship
Parent Container	-	Yes	Yes	-	Object	Scalar	Container
Containers	-	Yes	Yes	-	Object	Enum	Referential Containment Relationship
Version Descriptions	-	Yes	Yes	-	Object	Enum	Version Description
Current Of Series Count	-	Yes	Yes	-	Integer32	Scalar	

4.1.71.3 Description

The base class for all classes that may be versioned in a DMA system.

4.1.71.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Parent {dmaProp_Parent}**
Property Inherited from Containable
- **Parent Container {dmaProp_ParentContainer}**
Property Inherited from Containable
- **Containers {dmaProp_Containers}**
Property Inherited from Containable
- **Version Descriptions {dmaProp_VersionDescriptions}**

Property Attributes: Enum Of Objects of Class 'Version Description', System Generated, Read-Only

An enumeration of the Version Description objects by which this versionable object is referenced.

This property is only required if versioning is supported. If the object is not part of any version series, the enumeration is empty. Otherwise, it is a system-derived enumeration of those Version Descriptions which denote this object being part of a Version Series. This enumeration is manipulated by the `IdmaVersionable::SetCheckIn` method.

- **Current Of Series Count {dmaProp_CurrentOfSeriesCount}**

Property Attributes: Integer32, System Generated, Read-Only

Number of times this object is the current version of a version series

The value of this property is the number of version series for which this versionable object is the current version. The value of this property is affected by the Checkin method in up to two persistent objects:

- (1) The object that was formerly the current version (if any) has its value for this property decreased by one.
- (2) The Object that is being checked in has its value set to one (if the object is being made persistent for the first time) or increased by one (if the object is already persistent).

4.1.72 dmaClass_VersionDescription

Accounts for a single, specific occurrence of a versionable object in a Version Series.

Class ID: dmaClass_VersionDescription

Superclass: Containable

4.1.72.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]

4.1.72.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Version Description
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Parent	-	Yes	Yes	-	Object	Scalar	Direct Containment Relationship
Parent Container	-	Yes	Yes	-	Object	Scalar	Container
Containers	-	Yes	Yes	-	Object	Enum	Referential Containment Relationship
Version Series	Yes	Yes	Yes	Yes	Object	Scalar	Version Series
Is Current Version	Yes	Yes	Yes	Yes	Boolean	Scalar	
Version	Yes	Yes	Yes	-	Object	Scalar	Versionable
Branches	-	Yes	Yes	-	Object	Enum	Version Series

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Merges	-	Yes	Yes	-	Object	Enum	Version Series

4.1.72.3 Description

The version description object is used to store information describing the creation of a versionable object within a particular version series. Version Description objects are intended to carry any label information for identifying the connection, and indicating how the versionable object arrived at that point in the life cycle of the conceptual entity.

4.1.72.4 Property Descriptions

- **OIID {dmaProp_OIID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Parent {dmaProp_Parent}**
Property Inherited from Containable
- **Parent Container {dmaProp_ParentContainer}**
Property Inherited from Containable
- **Containers {dmaProp_Containers}**
Property Inherited from Containable
- **Version Series {dmaProp_VersionSeries}**

Property Attributes: Objects of Class 'Version Series', Property Must Be Supported, Value Required, System Generated, Read-Only

Indicates that version series this description is associated with (after check-in)

This property specifies the Version Series to which this description applies. That Version Series object will have this object in its Version Descriptions list.

- **Is Current Version {dmaProp_IsCurrentVersion}**

Property Attributes: Boolean, Property Must Be Supported, Value Required, System Generated, Read-Only

Is this the current version description for this series ?

This property describes whether this Version Description is the Current Version Description for the series specified in the Version Series property.

- **Version {dmaProp_Version}**

Property Attributes: Objects of Class 'Versionable', Property Must Be Supported, System Generated, Read-Only

The Versionable object this object describes.

This is the object, if any, that is the version of the conceptual entity represented by this Version Description. It can have a NULL value only when the versionable object that was supplied has been removed. The Versionable object SetCheckIn method produces Version Description objects with actual objects for the Version.

- **Branches {dmaProp_Branches}**

Property Attributes: Enum Of Objects of Class 'Version Series', System Generated, Read-Only

Property to support branched versioning. Not used in DMA 1.0.

For conceptual entities completely constructed using methods of the DMA 1.0 Versioning Model, this list is always empty.

- **Merges {dmaProp_Merges}**

Property Attributes: Enum Of Objects of Class 'Version Series', System Generated, Read-Only

Property to support merging of branches. Not used in DMA 1.0.

For conceptual entities completely constructed using methods of the DMA 1.0 Versioning Model, this list is always empty.

4.1.73 dmaClass_VersionSeries

The Version Series Class objects represent collections of versionable objects.

Class ID: dmaClass_VersionSeries

Superclass: Containable

4.1.73.1 Interfaces

- IUnknown
- IdmaObject
- IdmaProperties
- IdmaEditProperties [Optional]
- IdmaConnection [Optional]
- IdmaVersionSeries

4.1.73.2 Properties

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
OIID	-	Yes	Yes	-	String	Scalar	
Class Description	Yes	Yes	Yes	Yes	Object	Scalar	Class Description
This	-	Yes	Yes	Yes	Object	Scalar	Version Series
Create Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Update Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Delete Pending	-	Yes	Yes	Yes	Boolean	Scalar	
Parent	-	Yes	Yes	-	Object	Scalar	Direct Containment Relationship
Parent Container	-	Yes	Yes	-	Object	Scalar	Container
Containers	-	Yes	Yes	-	Object	Enum	Referential Containment Relationship
Configuration History	Yes	-	-	Yes	Object	Scalar	Configuration History
Is Primary Series	Yes	-	-	Yes	Boolean	Scalar	

Name	Impl. Req'd	System Gen'd	Read-Only	Value Req'd	Type	Cardinality	Required Class
Version Descriptions	Yes	Yes	Yes	-	Object	Enum	Version Description
Current Version Description	Yes	Yes	-	-	Object	Scalar	Version Description
Reservation	Yes	Yes	Yes	-	Object	Scalar	Reservation
Branched From Version Descriptions	-	Yes	Yes	-	Object	Enum	Version Description
Terminated Into Version Descriptions	-	Yes	Yes	-	Object	Enum	Version Description

4.1.73.3 Description

The version series is used to represent a collection of versionable objects that are versions of the same conceptual entity. The Version Series are made up of a linear sequence of zero or more Version Description objects and each Version Description object refers to at most one versionable object. The Version Series is part of exactly one Configuration History.

4.1.73.4 Property Descriptions

- **OID {dmaProp_OID}**
Property Inherited from DMA
- **Class Description {dmaProp_ClassDescription}**
Property Inherited from DMA
- **This {dmaProp_This}**
Property Inherited from DMA
- **Create Pending {dmaProp_CreatePending}**
Property Inherited from DMA
- **Update Pending {dmaProp_UpdatePending}**
Property Inherited from DMA
- **Delete Pending {dmaProp_DeletePending}**
Property Inherited from DMA
- **Parent {dmaProp_Parent}**
Property Inherited from Containable
- **Parent Container {dmaProp_ParentContainer}**
Property Inherited from Containable

- **Containers {dmaProp_Containers}**

Property Inherited from Containable

- **Configuration History {dmaProp_ConfigurationHistory}**

Property Attributes: Objects of Class 'Configuration History', Property Must Be Supported, Value Required

Configuration History associated with this object.

This object is the Configuration history to which this object belongs.

- **Is Primary Series {dmaProp_IsPrimarySeries}**

Property Attributes: Boolean, Property Must Be Supported, Value Required

This is the Primary Series under the Configuration History. This property must be set to make this Version Series the Primary Series under the associated Configuration History. At most one Version Series may be primary under a given Configuration History.

- **Version Descriptions {dmaProp_VersionDescriptions}**

Property Attributes: Enum Of Objects of Class 'Version Description', Property Must Be Supported, System Generated, Read-Only

An enumeration of the Version Description objects by which this versionable object is referenced. This property is only required if versioning is supported. If the object is not part of any version series, the enumeration is empty. Otherwise, it is a system-derived enumeration of those Version Descriptions which denote this object being part of a Version Series. This enumeration is manipulated by the IdmaVersionable::SetCheckIn method.

- **Current Version Description {dmaProp_CurrentVersionDescription}**

Property Attributes: Objects of Class 'Version Description', Property Must Be Supported, System Generated

Shortcut to latest description object in the Version Series. This property is a short-cut to find the latest Version Description in the Version Series. It is normally manipulated only by SetCheckIn, but can be set by direct manipulation if permitted by the implementation. This property is used by the SetCheckOutNext operation to determine what versionable object to clone.

- **Reservation {dmaProp_Reservation}**

Property Attributes: Objects of Class 'Reservation', Property Must Be Supported, System Generated, Read-Only

This property contains a Reservation object if there is an outstanding Checkout or Reservation against this Version Series. This is a system property -- it is manipulated by the SetCheckOutNext, SetReserveNext and SetRevoke methods of the Version Series object, and by the SetCheckIn method of a Versionable object which resolves the reservation.

- **Branched From Version Descriptions {dmaProp_BranchedFromVersionDescriptions}**

Property Attributes: Enum Of Objects of Class 'Version Description', System Generated, Read-Only

Enumeration of VersionDescriptions from other VersionSeries that this series springs from. Used to start a branch. This property is an enumeration of Version Descriptions from which this Version Series obtained its first version's source material. Applications which are DMA 1.0 compliant can safely ignore this property.

- **Terminated Into Version Descriptions {dmaProp_TerminatedIntoVersionDescriptions}**

Property Attributes: Enum Of Objects of Class 'Version Description', System Generated, Read-Only

Enumeration of Version Descriptions in other Version Series. Used to terminate a branch.

This property is an enumeration of Version Descriptions to which this Version Series contributed its last version's source material. This property is not used in DMA 1.0 implementations.

4.2 DMA Interfaces and Methods Reference

4.2.1 Introduction and Conventions

This chapter contains full descriptions of the DMA interfaces and methods. Each method description includes syntax, argument descriptions, method description, and return values.

The interface descriptions are ordered alphabetically, and within an interface the methods are ordered alphabetically.

Source code using DMA methods should contain `"#include <dmaiface.h>"`. Source code using DMA C-language functions should contain `"#include <dmacfunc.h>"`

A note on parameter names: DMA parameter names use a variant of Hungarian notation (see *Windows Programming* by Charles Petzold, 1992). The parameter names begin with a prefix that is a reminder of the type. The list of DMA types and their associated prefix is:

DMA_REFIID	riid
DmaBinaryValue *	p (pointer)
DmaBoolean	b (boolean)
DmaFloat64	d (double)
DmaIndex32	i (index)
DmaInt64	li
DmaInteger32	l (long)
Dmapv	pl (pointer to long)
DmaRC	rc
DmaUInteger32	ul (unsigned long)
pDmaBinary	p (pointer)
pDmaBoolean	pb (pointer to boolean)
pDmaDateTime	p
pDmaFloat64	pf (pointer to float)
pDmaId	p
pDmaIndex32	pi (index)
pDmaInteger32	pl
pDmapv	pp (pointer to pointer)
pDmaString	p
pDmaUInt64	puli (pointer to unsigned long)
pDmaUInteger32	pul
pp*	pp (pointer to pointer)

4.2.2 Free Standing Function and Entry-Points

This section describes the free standing functions and entry-points defined for the DMA specification.

4.2.2.1 DMA_GetServiceObject

4.2.2.1.1 Syntax

```
DmaRC DMA_GetServiceObject (
    pDmaVoid pvEnvInfo,
    pDmaString pProfile,
    Dmapv pIMalloc,
    Dmapv punkParent,
    DmaInteger32 lCharSetEncodingId,
    pDmaString pLocaleName,
    pDmaId pServiceObjectTypeId,
    pDmaId pServiceObjectId,
    DMA_REFIID riid,
    pDmapv ppIServiceObject)
```

4.2.2.1.2 Parameters

Name	Mode	Description
<i>pvEnvInfo</i>	input	(optional) Any environment-specific information required in order to locate the execution environment that the service object will operate within. The specific value is environment-specific. When <i>pvEnvInfo</i> is not defined for an environment or it is optional and omitted, NULL must be supplied. The specification of integration details for a given DMA environment will stipulate what is provided here for all <i>DMA_GetServiceObject</i> entries. It is typically, but not necessarily, the same <i>pvEnvInfo</i> supplied to <i>dmaConnectSystemManager</i> .
<i>pProfile</i>	input	(optional) Configuration-specific profile for the service object to be instantiated. Specified as part of registering the service object, this information is automatically supplied to the <i>GetSystemObject</i> function when an instance of the object is to be connected. This information is specific to the implementation of the object and has no significance in the DMA integration model beyond carrying it in the registry and supplying it to <i>GetSystemObject</i> . NULL will be supplied if this parameter was omitted in the registration.

Name	Mode	Description
<i>pIMalloc</i>	input	A interface to an object supporting the IMalloc interface. The service object must employ this IMalloc interface in constructing all storage structures delivered as output parameters from methods on the service object or any children of the service object. This applies any time that an interface delivers an output pointer that is not a pointer to an interface including pointers embedded in the data reached by such outputs (hereafter referred to as non-interface output pointers). The service object is required to use this pIMalloc for delivering of non-interface output pointers regardless of the memory allocation mechanism used for interfaces and in the internal operations of the object. The service object will also utilize the pIMalloc->Free method to free any non-interface output pointers it might obtain by invoking methods on any interface input parameters.
<i>punkParent</i>	input	An interface of the parent object. This is the object that is requesting the DMA_GetServiceObject operation
<i>ICharSetEncodingId</i>	input	The character set encoding id to be used by the returned service object and any objects created directly or indirectly via that interface.
<i>pLocaleName</i>	input	The locale to be used by the returned service object and any objects created directly or indirectly via the service object. If this parameter is NULL, the service object is to employ its default locale. If this parameter is non-NULL and does not correspond to a locale supported by this service object, the service object will utilize its default locale. The DMA String referenced by this parameter must utilize the character set encoding specified by the ICharSetEncodingId parameter otherwise this function will fail and return DMARC_BAD_PARAMETER.
<i>pServiceObjectTypeId</i>	input	The Dmald of the type of Service Object requested. In DMA 1.0 it should be an Id for either System, DocSpace or TextOrdering type.
<i>pServiceObjectId</i>	input	The Dmald of the object (e.g., System, DocSpace or TextOrdering) being instantiated. This is always supplied and provides for the same GetServiceObject being used to implement more than one object.

Name	Mode	Description
<i>pIMalloc</i>	input	A interface to an object supporting the IMalloc interface. The service object must employ this IMalloc interface in constructing all storage structures delivered as output parameters from methods on the service object or any children of the service object. This applies any time that an interface delivers an output pointer that is not a pointer to an interface including pointers embedded in the data reached by such outputs (hereafter referred to as non-interface output pointers). The service object is required to use this pIMalloc for delivering of non-interface output pointers regardless of the memory allocation mechanism used for interfaces and in the internal operations of the object. The service object will also utilize the pIMalloc->Free method to free any non-interface output pointers it might obtain by invoking methods on any interface input parameters.
<i>punkParent</i>	input	An interface of the parent object. This is the object that is requesting the DMA_GetServiceObject operation
<i>ICharSetEncodingId</i>	input	The character set encoding id to be used by the returned service object and any objects created directly or indirectly via that interface.
<i>pLocaleName</i>	input	The locale to be used by the returned service object and any objects created directly or indirectly via the service object. If this parameter is NULL, the service object is to employ its default locale. If this parameter is non-NULL and does not correspond to a locale supported by this service object, the service object will utilize its default locale. The DMA String referenced by this parameter must utilize the character set encoding specified by the ICharSetEncodingId parameter otherwise this function will fail and return DMARC_BAD_PARAMETER.
<i>pServiceObjectTypeId</i>	input	The Dmald of the type of Service Object requested. In DMA 1.0 it should be an Id for either System, DocSpace or TextOrdering type.
<i>pServiceObjectId</i>	input	The Dmald of the object (e.g., System, DocSpace or TextOrdering) being instantiated. This is always supplied and provides for the same GetServiceObject being used to implement more than one object.

Name	Mode	Description
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppIServiceObject</i>	output	Returns a reference to the requested interface for the service object, or NULL if unsuccessful.

4.2.2.1.3 Description

The implementation of a service object for use at a point-of-presence is accomplished by integrating a function named `DMA_GetServiceObject` of type `DMA_GetServiceObjectProc`. The function is packaged in a program in accordance with the requirements for the particular DMA point-of-presence environment. The program is in a form that can be registered via the `IdmaServiceRegistry` interface defined for that environment. The registration information is enough for an individual object to be instantiated as a DMA Object by bringing the "packaging" program to an operational state and invoking the `GetServiceObject` function.

It is important to understand that `GetServiceObject` is not a function that is called by DMA applications. `GetServiceObject` is a function that is delivered by the implementations of services integrated in the DMA point-of-presence. The program that carries the function is installed in a Service registry, and the System Manager and System Objects use the registry information to find DMA objects that have their implementations available at the point of presence.

Activation of a service object involves dynamic operation of the service object implementation. This operation is operating environment specific. The specification has been defined to be as generic as possible, limiting the differences between platforms to the specific semantics of certain parameters. Also, there are allowances for the service object to indicate to the point-of-presence how to locate and access the service object (via the `moduleLocation` and `moduleType` parameters of the registration mechanism).

4.2.2.1.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.2.2 dmaConnectSystemManager

4.2.2.2.1 Syntax

```
DmaRC dmaConnectSystemManager (  
    pDmaVoid pEnvInfo,  
    pDmaString pProfile,  
    Dmapv pIMalloc,  
    DmaInteger32 ICharSetEncodingId,  
    pDmaString pLocaleName,  
    DMA_REFIID riid,  
    pDmapv ppISysMgr)
```

4.2.2.2.2 Parameters

Name	Mode	Description
<i>pEnvInfo</i>	input	(optional) Any environment-specific information required in order to locate the execution environment that the system manager object will operate within. The specific value is environment-specific. When pEnvInfo is not defined for an environment or it is optional and omitted, NULL must be supplied.
<i>pProfile</i>	input	(optional) The information used to locate the system configuration information maintained by the system manager. When it is not required or optional, NULL must be supplied.

Name	Mode	Description
<i>pIMalloc</i>	input	(optional) A reference to an interface of an object that supplies the IMalloc interface. When this parameter is omitted, the System Manager provides an implementation that is used for this requirement throughout the life-time of the System Manager and any of its children. The System Manager and all of its children will employ this IMalloc interface in constructing all storage structures delivered as output parameters from methods on interfaces. This applies any time that an interface delivers an output pointer that is not a pointer to an interface, including pointers embedded in the data reached by such outputs (hereafter referred to as non-interface output pointers). The system manager and its children are required to use this pIMalloc for delivering non-interface output pointers regardless of the memory allocation mechanism used for interfaces themselves and in the internal operations of objects. (The proper way to release an interface pointer is always via the IUnknown::Release method of that interface.) In addition, any interfaces supplied as parameters to methods on children of the System Manager must deliver non-interface outputs that were allocated with this same IMalloc interface.
<i>ICharSetEncodingId</i>	input	The character set encoding id to be used by the returned system manager interface and any objects created directly or indirectly via that interface.
<i>pLocaleName</i>	input	(optional) The name of the locale to be used by the returned system manager interface and any system objects returned by the Idma-SystemManager::EnumerateSystems. If NULL, that value will be supplied in any SystemManager-synthesized ConnectSystem operations, as when performing EnumerateSystems. The DMA String referenced by this parameter must utilize the character set encoding specified by the ICharSetEncodingId parameter otherwise this function will fail and return DMARC_BAD_PARAMETER.
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppISysMgr</i>	output	Returns a reference to the requested interface for the system manager object, which will be NULL if unsuccessful.

4.2.2.2.3 Description

This is the first call that must occur when using the DMA Application Programming Interface. The DMA System Manager object is the starting point for access to all other DMA objects.

Returns an interface for a DMA System Manager object. The System Manager provides interfaces for enumerating the available DMA Systems, registering new DMA Systems, and connecting to a DMA System.

Each successful `dmaConnectSystemManager` procedure call returns a distinct System Manager object.

If the application employs other COM-based components, it is essential to use the same Memory Manager for delivering pointers to non-interface results everywhere. So long as this is done, interfaces from any of the COM-based components can be supplied to methods of any other interfaces, and there need be no concern about exactly which Memory Manager to use in releasing the storage of data returned from any interface method, no matter where the interface may have been obtained.

There are three simple rules of thumb:

- 1 If the program employs the Microsoft COM library (e.g., as part of OLE, MAPI, or other usage), the simplest way to provide an `IMalloc` to DMA is by first initializing COM and then using the Microsoft library `CoGetMalloc` to obtain the `IMalloc` to supply to `dmaConnectSystemManager`.
- 2 For all calls on `dmaConnectSystemManager` in a program, have exactly the same `pIMalloc` input on every call, `NULL` or otherwise.
- 3 For any other components that deliver method-allocated outputs via COM interfaces, arrange to use the same `pIMalloc` in those components as in (1-2) or vice versa.

This is sufficient to have all COM interfaces be interoperable without further concern.

Application development is also simplified considerably if the same character set encoding is specified for every `dmaConnectSystemManager` operation in the program.

At an abstract level, the information used by the system manager can be located anywhere. In practice the `pProfile` string will refer to a data source accessible within the local environment from which it is being used.

If the returned object does not support the interface specified by `riid`, the operation will fail and the result code `DMARC_BAD_INTERFACE` will be returned. In case the operation is successful, the returned interface must be released using the method `IUnknown::Release`.

There is a small well defined set of DMA return codes returned from this method. There is no DMA supplied mechanism for mapping DMA return codes into a meaningful message (e.g., `IdmaSystem::GetResultCodeDescription`), until a DMA System object becomes available.

4.2.2.2.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.3 IdmaAuthentication

Methods dealing with authenticating a user to a DMA system or document space.

This interface is implemented by a System or DocSpace object which requires authentication of users. If such authentication is not required, the interface need not be implemented.

4.2.3.1 IdmaAuthentication::AuthenticateUser

This method must be supported in every implementation of this interface.

4.2.3.1.1 Syntax

```
DmaRC IdmaAuthentication::AuthenticateUser (
    pDmaString pStringIn,
    ppDmaString ppStringOut)
```

4.2.3.1.2 Parameters

Name	Mode	Description
<i>pStringIn</i>	input	(optional) A pointer to a string containing the connection string, which is a set of keyword/value pairs.
<i>ppStringOut</i>	output	A pointer to a pDmaString string returned by the function. This string contains the connection string with additional keywords needed added to the pStringIn contents. If the connection was made, this pointer will be set to NULL

4.2.3.1.3 Description

This method is used to authenticate the user to the DMA System (via the typical ODBC BrowseConnect style dialog). The caller can test whether authentication is required by specifying a NULL value for pStringIn and testing for a return value of DMARC_ALREADY_AUTHENTICATED. This method may only be used to initially authenticate a user or test for whether a user is already authenticated, it cannot be used to change a user's authentication once set. Attempts to do so will return DMARC_ALREADY_AUTHENTICATED.

Other methods on a DMA system object may fail with a return code of DMARC_NOT_AUTHENTICATED if called before the user is authenticated via this method.

The DMA Connection Authentication mechanism is best described as a protocol. The client submits a request message (pStringIn) and the service responds with a response message (pointed to by pStringOut). Each of these messages is encoded in a DmaString and syntactically conforms to the following BNF syntax based upon the ODBC SQLBrowseConnect syntax:

```
request-message ::= connection-string
response-message ::= connection-string
connection-string ::= attribute[";" connection-string]
attribute ::= [ "*" ] keyword "=" value
keyword ::= dma-keyword | provider-keyword
dma-keyword ::= "SCHEME" | "UID" | "PWD" | "KRB-VERSION" |
    "KRB-PRINCIPAL" | "KRB-TICKET" | "KRB-AUTH" [ ":" localized-identifier ]
provider-keyword ::= identifier [ ":" localized-identifier ]
value ::= " " value-list " " | "?" | identifier
value-list ::= identifier [ ":" localized-identifier ]
    [ "," value-list]
```

identifier and localized-identifier consist of one or more characters. keyword is to be case insensitive. An asterisk preceding a keyword indicates that the keyword is optional and need not be present in the next request message.

4.2.3.1.4 DMA Authentication Schemes

The service response to an empty request message shall indicate which DMA authentication schemes the service supports. DMA 1.0 defines two authentication schemes and allows providers to extend the set of DMA authentication schemes. The DMA-BASIC scheme provides for a typical user-id and password based authentication. The DMA-KERBEROS scheme provides a mechanism for the client to provide the service with a Kerberos ticket and authenticator.

```
input connection string = ""
output connection string = "SCHEME=DMA-BASIC,
    DMA-KERBEROS,<other>"
return code = DMARC_NEED_MORE_DATA
```

Note: the service responds with one or more schemes. <other> denotes a scheme not standardized by DMA and must not start with the "DMA-" prefix.

4.2.3.1.5 DMA-BASIC Authentication

The Client selects the DMA-BASIC scheme:

```
input connection string = "SCHEME=DMA-BASIC"
output connection string = "SCHEME=DMA-BASIC;
    UID:UserName=?; PWD:Password=?"
return code = DMARC_NEED_MORE_DATA
```

- The Client provides userid and password:

```
input connection string = "SCHEME=DMA-BASIC;
    UID=Joe DMA;PWD=DMA4ME"
output connection string = NULL
return code = DMARC_SUCCESS
```

4.2.3.1.6 DMA-KERBEROS Authentication

The DMA-KERBEROS scheme supports different versions of Kerberos. A Kerberos scheme authentication message exchange follows:

- Client selects the DMA-KERBEROS scheme:

```
input connection string = "SCHEME=DMA-KERBEROS"
output connection string = "SCHEME=DMA-KERBEROS;
    KRB-VERSION=4,5,DCE"
return code = DMARC_NEED_MORE_DATA
```

- Client selects the DMA-KERBEROS version DCE:

```
input connection string = "SCHEME=DMA-KERBEROS;
    KRB-VERSION=DCE"
output connection string = "SCHEME=DMA-KERBEROS;
    KRB-VERSION=DCE;
    KRB-PRINCIPAL=DmaSystem@realm;
    KRB-AUTH=?;KRB-TICKET=?"
return code = DMARC_NEED_MORE_DATA
```

The response supplies the client with a Kerberos principal identifier via the KRB-PRINCIPAL value and requests that the client supply an appropriate Kerberos Authenticator and Ticket via the KRB-AUTH and KRB-TICKET values respectively.

- The Client responds:

```
input connection string = "SCHEME=DMA-KERBEROS;
    KRB-VERSION=DCE;
    KRB-PRINCIPAL=DmaSystem@realm;
    KRB-AUTH=<Base-64>;
    KRB-TICKET=<Base-64>"
output connection string = NULL
return code = DMARC_SUCCESS
```

where <Base-64> is a Base64 encoding per RFC 1521 of the Kerberos Authenticator and Ticket encrypted values.

4.2.3.1.7 Return Values

Name	Description
DMARC_ALREADY_AUTHENTICATED	The user is already authenticated.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NEED_MORE_DATA	(Warning) Indicates that more data is required to complete the browse connection operation.
DMARC_NETWORK_UNAVAILABLE	The network needed to perform this operation is not available.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_OK	(S_OK) Success.

4.2.3.2 IdmaAuthentication::AutoAuthenticateUser

Support for this method is optional.

4.2.3.2.1 Syntax

DmaRC **IdmaAuthentication::AutoAuthenticateUser** ()

4.2.3.2.2 Description

This method is used to request authentication without any caller supplied parameters. DMA System and Document Space implementations are not required to but may implement this behavior. DMA does not specify the implementation of the AutoAuthentication method, however such implementation may be predicated upon prior established authentication state or environmental state.

A client can prevent automatic authentication by not calling this method and instead utilize the IdmaAuthentication::Authenticate method to perform explicit authentication.

Other methods on a DMA system object may fail with a return code of DMARC_NOT_AUTHENTICATED if called before the user is authenticated via this method or the IdmaAuthentication::Authenticate method.

4.2.3.2.3 Return Values

Name	Description
DMARC_ALREADY_AUTHENTICATED	The user is already authenticated.
DMARC_NETWORK_UNAVAILABLE	The network needed to perform this operation is not available.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.

4.2.4 IdmaBatch

Methods for performing operations on a batch of scratchpad objects.

4.2.4.1 IdmaBatch::ExecuteChanges

This method must be supported in every implementation of this interface.

4.2.4.1.1 Syntax

DmaRC **IdmaBatch::ExecuteChanges** (
 Dmapv *pICallback*,
 Dmapv *pIListOfObj*,
 pDmaIndex32 *pIIndex*,
 DmaInteger32 *IProtectionLevel*,
 DmaBoolean *bRefreshFlag*)

4.2.4.1.2 Parameters

Name	Mode	Description
<i>pICallback</i>	input	(optional) Pointer to callback interface.
<i>pIListOfObj</i>	input	An interface pointer to a list object whose elements are the scratchpad objects that are to have their changes reflected in the database.
<i>pIIndex</i>	output	If the transaction fails, this parameter is set to the index of the object in the list that aborted the transaction. If the transaction succeeds, the value of this parameter is set to -1.
<i>IProtectionLevel</i>	input	Sets the protection level against concurrent updates. The value of this parameter must be one of DMA_MODIFY_PROTECTED or DMA_MODIFY_UNPROTECTED . The same level of protection applies to all scratchpad objects in the collection. The rules for this parameter are the same as for the ExecuteChange method of the IdmaConnection interface.
<i>bRefreshFlag</i>	input	If RefreshFlag is TRUE, the snapshots in the scratchpad objects are updated according to the rules for IdmaConnection::Refresh.

4.2.4.1.3 Description

The ExecuteChanges method is defined relative to the IdmaConnection:: ExecuteChange method. The most important difference between the two methods is that, instead of operating on a single object, ExecuteChanges operates on a list of objects as a batch. The ExecuteChanges method makes the changes

that ExecuteChange would have made to each object in the list. ExecuteChanges always performs all of the changes to the persistent store within a single transaction. If the change for any object fails, the entire transaction is aborted and backed out of the persistent store. If the change fails, none of the objects are refreshed, even if the bRefreshFlag is set to DMA_TRUE.

If there is any serial sequence of calls to IdmaConnection:: ExecuteChange on the objects in the pListOfObj list that would accomplish the desired final effect to the persistent store (assuming no concurrent updates to the persistent store), then the objects in the pListOfObj list are required to be in such an order, and ExecuteChanges applies the changes in order of increasing list index.

If there is no such sequence, then ExecuteChanges will attempt to make all the changes to the objects in increasing list index order. However, there may be transient inconsistent or illegal states during the execution of the transaction which can be tolerated as long as the final state is valid. If the final state is invalid, the transaction fails. For example, if the document space requires all documents to be contained directly, there is no series of ExecuteChange calls that will succeed in creating a new document. However, ExecuteChanges can succeed, if the document object is followed by the relationship object in the pListOfObj list.

ExecuteChanges does not affect any locks, except in the case that the current connection holds a write lock on a persistent object that ExecuteChanges deletes. In that case, the write lock is deleted as well.

Refer to the description of IdmaConnection:: ExecuteChange for the operation of the Create Pending, Delete Pending, and Update Pending properties, and for the effect of the protection level parameter.

In addition to the result codes listed below, the return code of this method can reflect any of the failures that can occur as a result of a create, update, or delete.

4.2.4.1.4 Return Values

Name	Description
DMARC_ABORT	The progress callback indicated that the method should be aborted.
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_REFERENCE	The object cannot be saved because it references another object which is not persistent.
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_FOREIGN_OBJECT	An object is not from the current document space.

Name	Description
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_MISSING_REFERENCE	The object cannot be saved because required reference to it is absent. For example, it may be required to be in a container and no appropriate relationship object exists.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.
DMARC_OK	(S_OK) Success.
DMARC_PROTECTION_LEVEL_NOT_SUPPORTED	The desired protection level is not supported by the doc space.
DMARC_READ_ONLY	Method failed because an object or property is read-only.
DMARC_REQUIRED_VALUE_ABSENT	A required property value has not been set.
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.

4.2.5 IdmaClassDescription

Class Descriptions are DMA objects that provide a meta-data description and a constructor for a class of DMA objects.

The IdmaClassDescription interface provides specialized methods supported by Class Description objects. The interface allows constructing an object of the described class, and testing if an object is in a class.

4.2.5.1 IdmaClassDescription::CreateInstance

Support for this method is optional.

4.2.5.1.1 Syntax

DmaRC **IdmaClassDescription::CreateInstance** (
DMA_REFIID *riid*,
pDmapv *ppIObject*)

4.2.5.1.2 Parameters

Name	Mode	Description
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppIObject</i>	output	Returns a reference to the requested interface for the object created, which will be NULL if unsuccessful.

4.2.5.1.3 Description

Creates a new transient object described by this Class Description object. The new object's properties will be assigned default values, and can be changed using the `IdmaEditProperties` interface. The new object may be made persistent using the `IdmaConnection::ExecuteChange` method.

If the returned object does not support the interface specified by `riid`, the operation will fail and the result code `DMARC_BAD_INTERFACE` will be returned. In case the operation is successful, the returned interface must be released using the method `IUnknown::Release`.

4.2.5.1.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.

4.2.5.2 IdmaClassDescription::IsOfClass

This method must be supported in every implementation of this interface.

4.2.5.2.1 Syntax

DmaRC IdmaClassDescription::IsOfClass (
pDmaId *pClassId*,
pDmaBoolean *pbIsOfClass*)

4.2.5.2.2 Parameters

Name	Mode	Description
<i>pClassId</i>	input	A pointer to the identifier of the class.
<i>pbIsOfClass</i>	output	Returns <code>DMA_TRUE</code> if the object conforms to the selected Class Id.

4.2.5.2.3 Description

Returns `DMA_TRUE` if and only if the described class conforms to the class specified by *pClassId*. To be conformant, the specified class ID must be an identifier for a class for which the described class is a subclass. All DMA Class Description objects should return `DMA_TRUE` if the specified class ID is `dmaClass_DMA`. (See also `IdmaObject::IsOfClass` for a similar method on any DMA object).

4.2.5.2.4 Return Values

Name	Description
DMARC_BAD_CLASSID	The supplied identifier does not reference an available class of object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.6 IdmaConnection

This interface provides access to the persistent state of an independently persistent scratchpad object. It exploits the logical connection to the document space in order to apply and remove locks, refresh the snapshot of the persistent state and to apply updates to the persistent state.

4.2.6.1 IdmaConnection::ApplyLock

Support for this method is optional.

4.2.6.1.1 Syntax

DmaRC **IdmaConnection::ApplyLock** (
DmaInteger32 *ILockType*)

4.2.6.1.2 Parameters

Name	Mode	Description
<i>ILockType</i>	input	The type of lock to be obtained on the object, one of DMA_LOCK_READ, DMA_LOCK_WRITE, or DMA_LOCK_EXIST.

4.2.6.1.3 Description

Lock an object in the persistent store in one of several modes. This permits client applications to operate while maintaining the proper level of data integrity on the object's persistent store. There is no requirement to use a lock in order to modify an object. If it is not possible to acquire the requested lock on the object, the status of the connections lock will not be altered, and a result code indicating the reason for failure will be returned. This function will not wait for a lock to become available, it will return as soon as it is determined that a lock can or cannot be acquired.

Valid values for *ILockType* include: DMA_LOCK_READ, DMA_LOCK_WRITE and DMA_LOCK_EXIST. DMA_LOCK_READ puts a read lock on this object. This prevents other connections from getting an write lock on the object. DMA_LOCK_WRITE puts an exclusive write lock on the object. This prevents other connections from getting a write lock or a read lock on the object. DMA_

LOCK_EXIST puts a existence lock on the object. The object cannot be deleted when an existence lock is held, even if deletion attempted by connection holding a write lock. This does not affect acquisition of read or write locks.

A lock is owned by a scratchpad object. A scratchpad object may own at most one lock on its related persistent object. If the scratchpad object already owns a lock of any type on its persistent object, the Lock method will return an error regardless of the value of the ILockType parameter, and the object's old lock will be unaffected.

A lock is removed following any of the events listed below:

- Invocation of IdmaConnection::RemoveLock on the same scratchpad object.
- The final IUnknown::Release invocation on the same scratchpad object.
- Termination of the logical document space connection via which the scratchpad object was obtained

4.2.6.1.4 Return Values

Name	Description
DMARC_BAD_LOCK_TYPE	The lock type is invalid.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OBJECT_DELETED	The object has been deleted since it was retrieved.
DMARC_OBJECT_LOCKED	The lock(s) on the persistent object prohibit the attempted operation.
DMARC_OBJECT_MODIFIED	The object has been modified since it was retrieved.
DMARC_OK	(S_OK) Success.

4.2.6.2 IdmaConnection::ExecuteChange

Support for this method is optional.

4.2.6.2.1 Syntax

DmaRC **IdmaConnection::ExecuteChange** (
 Dmapv *pICallback*,
 DmaInteger32 *IProtectionLevel*,
 DmaBoolean *bRefreshFlag*)

4.2.6.2.2 Parameters

Name	Mode	Description
<i>pICallback</i>	input	(optional) Pointer to callback interface
<i>IProtectionLevel</i>	input	Sets the protection level against concurrent updates. The value of this parameter must be one of DMA_MODIFY_UNPROTECTED, DMA_MODIFY_PROTECTED. See descriptive text below for details.
<i>bRefreshFlag</i>	input	If RefreshFlag is TRUE, the snapshot in the scratchpad object is updated according to the rules for IdmaConnection::Refresh.

4.2.6.2.3 Description

The ExecuteChange method makes the appropriate changes to the persistent object, whatever they may be: initially create a persistent object, update an existing persistent object, or delete an existing persistent object. The action is determined by the values of the three properties CreatePending, UpdatePending, and DeletePending.

The object is created if CreatePending is TRUE and DeletePending is FALSE.

The existing persistent object is modified if CreatePending is FALSE, UpdatePending is TRUE, and DeletePending is FALSE.

The existing object is deleted if CreatePending is FALSE, and DeletePending is TRUE.

If all three are FALSE, or if both CreatePending and DeletePending are TRUE, the method is a no-op as far as the document space persistent state is concerned, regardless of the value of IProtectionLevel. There shall be no updates to the persistent store as a result of such a no-op, but the bRefreshFlag parameter still has the usual effect on the scratchpad object state.

Here are details on the meaning of the value of the IProtectionLevel parameter. Each value of the protection level parameter is optionally supported. However, if the ExecuteChange method is supported, then at least one value of the protection level parameter must be supported.

For the creation case, the value of this parameter is irrelevant. The create will succeed, unless there is some unrelated problem, e.g., out of disk space, duplicate key value, etc.

For the deletion case, if the value of this parameter is DMA_MODIFY_PROTECTED, then the delete will fail if the persistent object does not exist and have the same property values as the original property values in the snapshot. The other values of this parameter have no effect on deletions.

For the case of updating an existing object, `ExecuteChange` fails with `DMARC_OBJECT_DELETED` if the independently persistent object to be updated does not exist in the persistent store, regardless of the value of the protection level parameter. Otherwise, when updating an existing independently persistent object, each value of the protection level parameter results in different behavior as follows:

- If the value of the protection level parameter is `DMA_MODIFY_UNPROTECTED`, then an attempt is made to update the independently persistent object in the persistent store to reflect the changes made to the scratchpad object. The implementation is free to assume that no concurrent updates have been made to the independently persistent object or any of its subobjects, providing only that the independently persistent object still exists. The updates are performed at the finest possible granularity and do not affect any part of the persistent state which is unmodified in the scratchpad object. In the case of lists, only the minimal insertions, deletions, and replacements need be performed. Thus, for example, if a property or list element was deleted from the scratchpad object, it will be deleted from the persistent object, and none of the rest of the state of the persistent object will be altered. In the case of dependently persistent subobjects, this is recursively true. For purposes of this paragraph, "subobject" includes list elements. If any changes to the persistent object have been made in the persistent store that conflict with any changes made to the scratchpad, then the result is undefined. Some modifications may no longer be applicable, and, if this situation is detected, these modifications may be quietly discarded or an error may be returned, at the option of the implementation. Since this value of the protection level parameter provides no protection against concurrent updates to the persistent store, it is recommended that locking be used to protect against concurrent updates (if locking is supported), or, even better, that the protection level value `DMA_MODIFY_PROTECTED` be used instead (if it is supported).
- If the value of the protection level parameter is `DMA_MODIFY_PROTECTED`, the result is effectively the same as if an atomic combination of `IsCurrent` and `DMA_MODIFY_UNPROTECTED` were performed: If the independently persistent object is not current in the sense of `IsCurrent`, then the update fails with `DMARC_OBJECT_MODIFIED`. Otherwise, it is known that neither the independently persistent object nor any of its dependently persistent subobjects has changed. In that case, the changes made to the scratchpad are applied to the persistent object. The updates are performed at the finest possible granularity and do not affect any part of the state of the persistent object which is unmodified in the scratchpad. It is recommended that clients use this value of the protection level parameter, if it is supported, in preference to using `DMA_MODIFY_UNPROTECTED`. This value of the protection level parameter provides concurrency protection when updating an object while avoiding the additional overhead that would be incurred by use of the locking and unlocking methods (assuming locking is supported)."

Many more return values are possible from this method than are listed below. They are detailed in individual method descriptions -- see especially any method with "Set" in its name.

4.2.6.2.4 Return Values

Name	Description
DMARC_ABORT	The progress callback indicated that the method should be aborted.
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_REFERENCE	The object cannot be saved because it references another object which is not persistent.
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_MISSING_REFERENCE	The object cannot be saved because required reference to it is absent. For example, it may be required to be in a container and no appropriate relationship object exists.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.
DMARC_OBJECT_DELETED	The object has been deleted since it was retrieved.
DMARC_OBJECT_LOCKED	The lock(s) on the persistent object prohibit the attempted operation.
DMARC_OBJECT_MODIFIED	The object has been modified since it was retrieved.
DMARC_OK	(S_OK) Success.
DMARC_PROTECTION_LEVEL_NOT_SUPPORTED	The desired protection level is not supported by the doc space.
DMARC_READ_ONLY	Method failed because an object or property is read-only.
DMARC_REQUIRED_VALUE_ABSENT	A required property value has not been set.
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.

4.2.6.3 IdmaConnection::ExecuteRefresh

Support for this method is optional.

4.2.6.3.1 Syntax

DmaRC **IdmaConnection::ExecuteRefresh** ()

4.2.6.3.2 Description

Refreshes the scratchpad object with an up to date snapshot from the persistent store. If successful, any changes made to the scratchpad object prior to this call will be discarded.

If the persistent object no longer exists, the error DMARC_OBJECT_DELETED is returned, and the scratchpad object is unchanged.

If the persistent object exists and is successfully retrieved, the CreatePending, UpdatePending, and DeletePending properties are set to FALSE, FALSE, FALSE. All scalar properties of the scratchpad object are set to match the current state of the persistent store, with the usual consistency guarantees. All object-valued properties are reset to the unbound state.

4.2.6.3.3 Return Values

Name	Description
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OBJECT_DELETED	The object has been deleted since it was retrieved.
DMARC_OBJECT_LOCKED	The lock(s) on the persistent object prohibit the attempted operation.
DMARC_OK	(S_OK) Success.
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.

4.2.6.4 IdmaConnection::IsCurrent

Support for this method is optional.

4.2.6.4.1 Syntax

DmaRC **IdmaConnection::IsCurrent** (
pDmaBoolean *pblsCurrent*)

4.2.6.4.2 Parameters

Name	Mode	Description
<i>pblsCurrent</i>	output	Returns DMA_TRUE if the current state of the object's snapshot is current with the persistent state of the object.

4.2.6.4.3 Description

Returns DMA_TRUE if the persistent state of the object has not been changed since the IdmaConnection interface was delivered, or the last call to IdmaConnection::ExecuteRefresh, whichever is later.

The method may be conservative and return DMA_FALSE for the value of *pbIsCurrent if there is good reason to believe that the independently persistent object or one of its dependently persistent subobjects may have changed, but determining with certainty whether the object has actually changed is considered to be too expensive. For example, a difference in the value of a change count or epoch field in the independently persistent object is considered to be a sufficiently good reason to believe that the object may have changed. It is not necessary to actually check whether the old and new values of all properties and all other parts of the persistent state have changed. (For example, that might be considered too expensive in the case of content elements and large binary properties.) Note, therefore, that DMA_FALSE might be returned for *pbIsCurrent, even though a property was set to its original value, resulting in no net change. In contrast, being pessimistic and always returning DMA_FALSE for *pbIsCurrent is not acceptable.

4.2.6.4.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.

4.2.6.5 IdmaConnection::RemoveLock

Support for this method is optional.

4.2.6.5.1 Syntax

DmaRC IdmaConnection::RemoveLock ()

4.2.6.5.2 Description

Remove the lock that is held by this connection on the persistent object, making it available to other connections. If there is no lock held on this connection the return code DMARC_NOT_LOCKED will be returned.

4.2.6.5.3 Return Values

Name	Description
DMARC_NOT_LOCKED	The object is not locked.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.

4.2.6.6 IdmaConnection::SetDeletePending

Support for this method is optional.

4.2.6.6.1 Syntax

DmaRC **IdmaConnection::SetDeletePending** (
DmaBoolean *bDeleteFlag*)

4.2.6.6.2 Parameters

Name	Mode	Description
<i>bDeleteFlag</i>	input	DMA_TRUE if object is to be marked for delete, DMA_FALSE to be unmarked for delete.

4.2.6.6.3 Description

The value of the DeletePending property is set in the current scratchpad object. This preconditions it so that if the DeletePending property is TRUE, IdmaConnection::ExecuteChange or IdmaBatch::ExecuteChanges will delete the persistent object.

Successful operation of this method does not depend on the value of the Delete Pending property.

4.2.6.6.4 Return Values

Name	Description
DMARC_CONFLICTING_OPERATION	The call on this primary intentional method logically conflicts with a prior call on a different primary intentional method.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.

4.2.6.6.5 Deferred Return Values

Name	Description
DMARC_OBJECT_REFERENCED	The object cannot be deleted because it is referenced by other objects.
DMARC_REFERENCES_OTHERS	The object cannot be deleted because it holds references to other objects.

4.2.7 IdmaContentTransfer

This interface provides access to content data (represented by objects of `dmaClass_ContentTransfer`) which is directly captured and managed by a document space.

4.2.7.1 IdmaContentTransfer::CopyToResource

Support for this method is optional.

4.2.7.1.1 Syntax

`DmaRC IdmaContentTransfer::CopyToResource (`
`pDmaString pDestName,`
`DmaInteger32 flags)`

4.2.7.1.2 Parameters

Name	Mode	Description
<i>pDestName</i>	input	A string that contains a locally accessible resource name in URL format.
<i>flags</i>	input	Flags that modify creation of the file

4.2.7.1.3 Description

Makes a copy of the captured data in a resource of the specified name. This method is optional on already persistent content transfer elements and should return `DMARC_NOT_SUPPORTED` on a content transfer element which has not yet been saved.

If the copy occurs successfully the client is responsible for removing the resource when it is no longer needed. The resource is owned by the client, and the content transfer element will retain no memory of its existence.

The flags parameter is a bitwise OR of any of the following integer values:

- `DMA_CREATE_REPLACE_EXISTING`

This flag dictates behaviour if the resource named by `pDestName` already exists at the time this method is called. If the flag is set, the resource should be replaced with the source material. If this flag is not set, the method leaves the existing resource intact and returns `DMARC_NO_ACCESS`. This flag is ignored if the specified resource does not yet exist.

- `DMA_CREATE_PRESERVE_DATETIME`

Specifies how the creation time of the resource should be set. If the flag is set and the document space has recorded the creation time of incoming document content, the creation time of the new resource is set to original

creation time of the document content. If the flag is not set, the creation time for the resource should be set according to platform defaults. (Typically, to the current time).

For a document space which supports content replacement, a call to `SetCaptureStream` or `SetCaptureResource` causes subsequent calls to this method to behave as for the new case, which is to return `DMARC_NOT_SUPPORTED`. In other words, the existing content rendered inaccessible through this content element instance.

It is not required that a document space which supports this method support all possible forms of URL. However, at minimum the file: form should be supported.

If the URL supplied includes security credentials, these should be used to access the resource in order to complete the data transfer. Otherwise, the operation must be performed using the security context of the calling thread. This should hold irrespective of whether the service provider operates in a separate process with its own security context.

Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_URL	The resource URL is not syntactically valid.
DMARC_DEVICE_ERROR	An error has occurred reading or writing a hardware device.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_URL_PROTOCOL_NOT_SUPPORTED	The URL protocol specified in the reference is not supported.

4.2.7.2 `IdmaContentTransfer::GetResourceName`

Support for this method is optional.

4.2.7.2.1 Syntax

`DmaRC IdmaContentTransfer::GetResourceName (`
`ppDmaString ppName)`

4.2.7.2.2 Parameters

Name	Mode	Description
<i>ppName</i>	output	Returns a resource name in URL format.

4.2.7.2.3 Description

This method behaves differently according to whether it is applied to a new (unsaved) content element or to an existing persistent content element.

In the former case, this method provides a way to interrogate the capture mode for the object. If resource name mode has been specified, the method will return a copy of the URL supplied to `SetCaptureResource`. If stream mode has been specified or no mode has yet been set, the method will return `DMARC_VALUE_NOT_SET`.

In the latter case this method returns a resource name that the client application may be able use to directly access the captured content data.

This methods makes no guarantee that the application will in fact be able to access the content data via the resource name returned, since this depends on the protocol specified in the URL and on the security which is applied to it.

For a docspace which supports content replacement, a call to `SetCaptureStream` or `SetCaptureResource` causes subsequent calls to this method to behave as for the new case. In other words, the existing content is rendered inaccessible through this content element instance.

This method must be supported if `SetCaptureResource` is supported; otherwise support is optional.

4.2.7.2.4 Return Values

Name	Description
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.7.3 IdmaContentTransfer::GetStream

This method must be supported in every implementation of this interface.

4.2.7.3.1 Syntax

```
DmaRC IdmaContentTransfer::GetStream (
    DMA_REFIID riid,
    pDmapv ppIStream)
```

4.2.7.3.2 Parameters

Name	Mode	Description
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>pplStream</i>	output	Returns a stream object that can deliver the captured data.

4.2.7.3.3 Description

This method behaves differently according to whether it is applied to a new (unsaved) content element or to an existing persistent content element.

In the former case, this method provides a way to interrogate the capture mode for the object. If stream mode has been specified, the method will return a reference to the stream object supplied to SetCaptureStream. If resource name mode has been specified or no mode has yet been set, the method will return DMARC_VALUE_NOT_SET.

In the latter case, the method returns a stream object providing direct access to the captured content data.

In both cases, the method exhibits "by reference" behaviour. A second or subsequent call returns a reference to the same object as the first call. In both cases, the position of the seek pointer on the stream object is undefined.

For a document space which supports content replacement, a call to SetCaptureStream or SetCaptureResource causes subsequent calls to this method to behave as for the new case. In other words, the existing content is rendered inaccessible through this content element instance.

4.2.7.3.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.7.4 IdmaContentTransfer::SetCaptureResource

Support for this method is optional.

4.2.7.4.1 Syntax

DmaRC IdmaContentTransfer::SetCaptureResource (
pDmaString pResourceName)

4.2.7.4.2 Parameters

Name	Mode	Description
<i>pResourceName</i>	input	A URL format string specifying the name of the resource containing the content data which is to be captured.

4.2.7.4.3 Description

Requests content capture from a named resource (aka "by filename").

This method is optional. Where supported, it may be so only for new (unsaved) content transfer element objects or for already saved content transfer elements (allowing for content replacement).

It is strongly recommended that the document space validate the resource name at the time of this method invocation, to determine that it is of a supported syntax and refers to an existing resource. It is not however recommended that the document space attempt to place a lock of some kind on the resource in an attempt to ensure its availability through to the time the content element is saved, since this may be expensive and may interfere with other legitimate operations which take place in the interim. It is therefore the case that (re)validation must be performed at the time the content element is made persistent.

It is not required that a document space which supports this method support all possible forms of URL. However, at minimum the file: form should be supported.

If the client application does not set a value for the Retrieval Name property of the content transfer element object to which this method is applied, the implementation is permitted to parse a suitable value out of *pResourceName*. However, this should not become visible as the property value until after the content element is made persistent.

If the URL specified includes security credentials, these should be used to access the content data. Otherwise, access to the content data must be attempted under the security context of the client thread which called this method effective at the time of the call, and the ExecuteChange operation should fail if this is not sufficient. This should hold irrespective of whether the service provider operates in a separate process with its own security context.

4.2.7.4.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.

Name	Description
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.
DMARC_URL_PROTOCOL_NOT_SUPPORTED	The URL protocol specified in the reference is not supported.

4.2.7.4.5 Deferred Return Values

Name	Description
DMARC_BAD_URL	The resource URL is not syntactically valid.
DMARC_DEVICE_ERROR	An error has occurred reading or writing a hardware device.
DMARC_RESOURCE_NOT_FOUND	The indicated resource was not found.

4.2.7.5 IdmaContentTransfer::SetCaptureStream

This method must be supported in every implementation of this interface.

4.2.7.5.1 Syntax

DmaRC **IdmaContentTransfer::SetCaptureStream** (
Dmapv *pStream*)

4.2.7.5.2 Parameters

Name	Mode	Description
<i>pStream</i>	input	An interface pointer to an object capable of providing an IdmaStream interface.

4.2.7.5.3 Description

Requests content capture from a stream and specifies the stream from which to capture. This method must be supported on a new (not yet saved) content transfer element by all document space implementations. A document space which supports update of captured content may also allow this on a previously saved content transfer element as a way of effecting content replacement.

This method may be invoked an arbitrary number of times, in arbitrary combination with calls to SetCaptureResource on the same content transfer object. The ultimate effect when this object is saved is as if only the most recent call had been issued.

The stream supplied must be positioned at the beginning of the data to be captured. The implementation must capture the data using only the ReadStreamData method of the stream, since the stream is not required to support arbitrary positioning.

4.2.7.5.4 Return Values

Name	Description
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.7.5.5 Deferred Return Values

Name	Description
DMARC_DEVICE_ERROR	An error has occurred reading or writing a hardware device.

4.2.8 IdmaDocSpace

This is the specialized interface for the DMA Document Space object. It provides methods for construction of new objects in the Document Space and for manipulating objects that are managed by the Document Space.

4.2.8.1 IdmaDocSpace::ConnectAndLockObject

Support for this method is optional.

4.2.8.1.1 Syntax

DmaRC **IdmaDocSpace::ConnectAndLockObject** (
 pDmaString *pObjectId*,
 DmaInteger32 *ILockType*,
 DMA_REFIID *riid*,
 pDmapv *ppIObject*)

4.2.8.1.2 Parameters

Name	Mode	Description
<i>pObjectId</i>	input	A pointer to the globally unique object instance identifier (OIID) of the persistent object to be retrieved.
<i>ILockType</i>	input	The type of lock to be obtained on the object. One of DMA_LOCK_READ, DMA_LOCK_WRITE, or DMA_LOCK_EXIST.
<i>riid</i>	input	Interface Identifier for the desired interface on the resulting object.
<i>ppIObject</i>	output	An interface on the object created, which will be NULL if the method fails.

4.2.8.1.3 Description

ConnectAndLockObject is the same as ConnectObject, except that a lock is atomically placed on the object connected to if the method succeeds. This method is supported if and only if the locking methods (i.e., ApplyLock and RemoveLock) are supported.

4.2.8.1.4 Return Values

Name	Description
DMARC_ACCESS_DENIED	(E_ACCESSDENIED) The requester has insufficient access rights to perform the requested operation.
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_LOCK_TYPE	The lock type is invalid.
DMARC_BAD_OIID	The supplied buffer does not contain a valid OIID.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_LOST_CONNECTION	The logical connection of the current object to the persistent store has been lost permanently. The operation could not be completed.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_NOT_FOUND	Requested item not found.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OBJECT_LOCKED	The lock(s) on the persistent object prohibit the attempted operation.
DMARC_OK	(S_OK) Success.

4.2.8.2 IdmaDocSpace::ConnectObject

Support for this method is optional.

4.2.8.2.1 Syntax

```
DmaRC IdmaDocSpace::ConnectObject (
    pDmaString pObjectId,
    DMA_REFIID riid,
    pDmapv ppIObject)
```

4.2.8.2.2 Parameters

Name	Mode	Description
<i>pObjectId</i>	input	A pointer to the globally unique object instance identifier (OIID) of the persistent object to be retrieved. This identifier may be known by the caller or obtained from previously instantiated objects.
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppIObject</i>	output	An interface on the object created, which will be NULL if unsuccessful on an initial open.

4.2.8.2.3 Description

ConnectObject makes a scratchpad copy in the client's process space of the referenced persistent object (i.e. it instantiates an object that is already persistent). A snapshot of the object's properties will be set from the persistent store; they may be changed using the IdmaEditProperties interface. Changes can be made persistent using the IdmaConnection::ExecuteChange method.

At any time the object can be checked against the persistent copy for consistency using the IdmaConnection::IsCurrent call. If the object has changed, or the caller simply wants the properties refreshed, the IdmaConnection::ExecuteRefresh call may be used. That method will not instantiate a new object, but will retrieve a new copy of the persistent information into the existing instance.

If the persistent object has been locked for exclusive use by another connection, the ConnectObject method will not be successful.

If the returned object does not support the interface specified by riid, the operation will fail and the result code DMARC_BAD_INTERFACE will be returned. In case the operation is successful, the returned interface must be released using the method IUnknown::Release.

If pObjectId does not refer to an existing persistent object, DMARC_NOT_FOUND will be returned.

This method fails with the error DMARC_NOT_AUTHENTICATED if the document space is not in an authenticated state.

4.2.8.2.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_OIID	The supplied buffer does not contain a valid OIID.

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_NOT_FOUND	Requested item not found.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OBJECT_LOCKED	The lock(s) on the persistent object prohibit the attempted operation.
DMARC_OK	(S_OK) Success.
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.

4.2.8.3 IdmaDocSpace::ExecuteDisconnect

This method must be supported in every implementation of this interface.

4.2.8.3.1 Syntax

DmaRC **IdmaDocSpace::ExecuteDisconnect** ()

4.2.8.3.2 Description

Terminates the logical connection to the underlying document repository. This may be useful to free up connection resources in environments in which direct control cannot be exercised over the lifetime of the DocSpace COM object and objects obtained from it.

Once this method is executed successfully, other methods (on the DocSpace object or objects obtained from it) that need the connection to function will return DMARC_DISCONNECTED.

Note that if the DocSpace object and all objects generated from it are released, the connection is implicitly closed. ExecuteDisconnect() just provides a way of doing this explicitly. Once a DocSpace object is disconnected, it cannot be reconnected.

4.2.8.3.3 Return Values

Name	Description
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_OK	(S_OK) Success.

4.2.8.4 IdmaDocSpace::GetScope

Support for this method is optional.

4.2.8.4.1 Syntax

```
DmaRC IdmaDocSpace::GetScope (
DMA_REFIID riid,
pDmapv ppIScope)
```

4.2.8.4.2 Parameters

Name	Mode	Description
<i>riid</i>	input	Interface Identifier for the desired interface on resulting Document Space object.
<i>ppIScope</i>	output	Returns the selected interface for the merged scope object.

4.2.8.4.3 Description

Returns a Scope Object for the Document Space. The scope supplies the meta-data used to formulate the parameters for a query operation and to execute a query operation.

Cross document space queries can be performed by placing document space scope objects on an object list and using the `IdmaScopeFactory::CreateScope` function to create a merged scope. The merged scope can be used to formulate and execute queries coordinated across multiple document spaces.

If the returned object does not support the interface specified by *riid*, the operation will fail and the result code `DMARC_BAD_INTERFACE` will be returned. In case the operation is successful, the returned interface must be released using the method `IUnknown::Release`.

This method fails with the error `DMARC_NOT_AUTHENTICATED` if the document space is not in an authenticated state.

4.2.8.4.4 Return Values

Name	Description
<code>DMARC_BAD_INTERFACE</code>	(<code>E_NOINTERFACE</code>) The requested interface is not supported by this object.
<code>DMARC_BAD_PARAMETER</code>	(<code>E_INVALIDARG</code>) Invalid input parameter.
<code>DMARC_NOT_AUTHENTICATED</code>	The user is not authenticated.
<code>DMARC_NOT_SUPPORTED</code>	This method is not supported in the context of this session or object.
<code>DMARC_OK</code>	(<code>S_OK</code>) Success.

4.2.9 IdmaEditList

Lists are a fundamental feature in managing documents, so there are several operations available for manipulating them and the objects contained within them. Items within a list are kept in inserted order and can be identified by an index (ordinal position).

The methods used to access a DMA List are segmented into two groups of interfaces, the methods that allow readonly access to the list (IdmaList), and the methods that allow modification of the list (IdmaEditList). The IdmaEditList interface is only available on list objects that are editable. For each list datatype, there is a specific interface that offers methods particular to that datatype, as well as all the methods of IdmaEditList, (e.g., IdmaEditListOfString).

4.2.9.1 IdmaEditList::DeleteElement

This method must be supported in every implementation of this interface.

4.2.9.1.1 Syntax

DmaRC **IdmaEditList::DeleteElement** (
DmaIndex32 *iIndex*)

4.2.9.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list of the element to delete.

4.2.9.1.3 Description

This method deletes an item from a list based on an ordinal position, provided by the caller. All following list elements will have their ordinal position decremented by one. Permitted values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

Items on the list are freed as appropriate for their type.

4.2.9.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.9.1.5 Deferred Return Values

Name	Description
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.

4.2.9.2 IdmaEditList::TruncateList

This method must be supported in every implementation of this interface.

4.2.9.2.1 Syntax

DmaRC **IdmaEditList::TruncateList** (
DmaIndex32 *iIndex*)

4.2.9.2.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list which will be the first item deleted. All following items will also be deleted.

4.2.9.2.3 Description

This method will delete all items from the list starting with the item at the supplied ordinal position. All subsequent items in the list will also be deleted. Items on the list will be released as appropriate for their type.

Examples: An index value of zero (0) will result in the list being emptied. An index value of one (1) will result in all but the first item in the list being deleted. An index of $n-1$, where n is the number of items in the list, will result in the last item on the list being deleted.

4.2.9.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.9.2.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.

4.2.10 IdmaEditListOfBinary

This interface provides the methods for modification of lists that contain buffers of binary data.

This interface offers all the methods of the IdmaEditList interface, as well as those listed below.

4.2.10.1 IdmaEditListOfBinary::InsertBinary

This method must be supported in every implementation of this interface.

4.2.10.1.1 Syntax

```
DmaRC IdmaEditListOfBinary::InsertBinary (
    DmaIndex32 iIndex,
    DmaBinaryValue *pBinaryValue)
```

4.2.10.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the object will be placed.
<i>pBinaryValue</i>	input	A pointer to the binary bytes to be inserted into the list.

4.2.10.1.3 Description

This function inserts a binary value into the list given on an ordinal position provided by the caller. All list elements that have indexes greater than *iIndex* at the time of the call will have their index in the list incremented by one. Permitted values for *iIndex* are 0 through *n*, where *n* is the number of elements in the list at the time of the call. A value of *n* appends a new value to the end of the list.

4.2.10.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.10.1.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.10.2 IdmaEditListOfBinary::ReplaceBinary

This method must be supported in every implementation of this interface.

4.2.10.2.1 Syntax

```
DmaRC IdmaEditListOfBinary::ReplaceBinary (
    DmaIndex32 iIndex,
    DmaBinaryValue *pBinaryValue)
```

4.2.10.2.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the object will be placed.
<i>pBinaryValue</i>	input	A pointer to the binary bytes that will replace the item at the ordinal position in the list.

4.2.10.2.3 Description

Replaces a binary value in a list with the supplied binary value given an ordinal position value as input. Valid values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

4.2.10.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.10.2.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.11 IdmaEditListOfBoolean

This interface provides methods for modification of lists that contain boolean values.

This interface offers all the methods of the IdmaEditList interface, as well as those listed below.

4.2.11.1 IdmaEditListOfBoolean::InsertBoolean

This method must be supported in every implementation of this interface.

4.2.11.1.1 Syntax

DmaRC **IdmaEditListOfBoolean::InsertBoolean** (
DmaIndex32 *iIndex*,
DmaBoolean *bBooleanValue*)

4.2.11.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>bBooleanValue</i>	input	The boolean value that will be inserted into the list.

4.2.11.1.3 Description

This function inserts a boolean value into the list given on an ordinal position provided by the caller. All list elements that have indexes greater than *iIndex* at the time of the call will have their index incremented by one. Permitted values for *iIndex* are 0 through *n*, where *n* is the number of elements in the list at the time of the call. A value of *n* appends a new value to the end of the list.

4.2.11.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.

Name	Description
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.11.1.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.11.2 IdmaEditListOfBoolean::ReplaceBoolean

This method must be supported in every implementation of this interface.

4.2.11.2.1 Syntax

DmaRC **IdmaEditListOfBoolean::ReplaceBoolean** (
DmaIndex32 *iIndex*,
DmaBoolean *bBooleanValue*)

4.2.11.2.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>bBooleanValue</i>	input	The new boolean value that will replace the value in the list.

4.2.11.2.3 Description

Replaces a boolean value in a list with the supplied boolean value given an ordinal position value as input. Valid values for *iIndex* are 0 through $n-1$, where n is the number of elements in the list at the time of the call.

4.2.11.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.

Name	Description
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.11.2.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.12 IdmaEditListOfDateTime

This interface provides methods for modification of lists that contain dates.

This interface offers all the methods of the IdmaEditList interface, as well as those listed below.

4.2.12.1 IdmaEditListOfDateTime::InsertDateTime

This method must be supported in every implementation of this interface.

4.2.12.1.1 Syntax

```
DmaRC IdmaEditListOfDateTime::InsertDateTime (
    DmaIndex32 ilIndex,
    pDmaDateTime pDateTimeValue)
```

4.2.12.1.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>pDateTimeValue</i>	input	The DateTime value that will be inserted into the list.

4.2.12.1.3 Description

This function inserts a DateTime value into the list given on an ordinal position provided by the caller. All list elements that have indexes greater than *ilIndex* at the time of the call will have their index in the list incremented by one. Permitted values for *ilIndex* are 0 through *n*, where *n* is the number of elements in the list at the time of the call. A value of *n* appends a new value to the end of the list.

4.2.12.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.12.1.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.12.2 IdmaEditListOfDateTime::ReplaceDateTime

This method must be supported in every implementation of this interface.

4.2.12.2.1 Syntax

DmaRC **IdmaEditListOfDateTime::ReplaceDateTime** (
DmaIndex32 *iIndex*,
pDmaDateTime *pDateTimeValue*)

4.2.12.2.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the value will be replaced.
<i>pDateTimeValue</i>	input	The new DateTime value that will replace the value in the list.

4.2.12.2.3 Description

Replaces a DateTime value in a list with the supplied DateTime value given an ordinal position value as input. Valid values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

4.2.12.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.12.2.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.13 IdmaEditListOfFloat64

This interface provides methods for modifying lists that contain 64 bit float values.

This interface offers all the methods of the IdmaEditList interface, as well as those listed below.

4.2.13.1 IdmaEditListOfFloat64::InsertFloat64

This method must be supported in every implementation of this interface.

4.2.13.1.1 Syntax

DmaRC **IdmaEditListOfFloat64::InsertFloat64** (
DmaIndex32 *iIndex*,
DmaFloat64 *fFloatValue*)

4.2.13.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>fFloatValue</i>	input	The float value that will be inserted into the list.

4.2.13.1.3 Description

This function inserts a float value into the list given on an ordinal position provided by the caller. All list elements that have indexes greater than *iIndex* at the time of the call will have their index in the list incremented by one. Permitted values for *iIndex* are 0 through *n*, where *n* is the number of elements in the list at the time of the call. A value of *n* appends a new value to the end of the list.

4.2.13.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.13.1.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.13.2 IdmaEditListOfFloat64::ReplaceFloat64

This method must be supported in every implementation of this interface.

4.2.13.2.1 Syntax

```
DmaRC IdmaEditListOfFloat64::ReplaceFloat64 (
    DmaIndex32 iIndex,
    DmaFloat64 fFloatValue)
```

4.2.13.2.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>fFloatValue</i>	input	The new float value that will replace the value in the list.

4.2.13.2.3 Description

Replaces a float value in a list with the supplied float value given an ordinal position value. Valid values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

4.2.13.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.13.2.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.14 IdmaEditListOfId

This interface provides methods for modification of lists that contain DMA Identifiers.

This interface offers all the methods of the IdmaEditList interface, as well as those listed below.

4.2.14.1 IdmaEditListOfId::InsertId

This method must be supported in every implementation of this interface.

4.2.14.1.1 Syntax

```
DmaRC IdmaEditListOfId::InsertId (
    DmaIndex 32iIndex,
    pDmaId pIdValue)
```


4.2.14.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>pldValue</i>	input	The Id value that will be inserted into the list.

4.2.14.1.3 Description

This function inserts a DMA identifier value into the list given on an ordinal position provided by the caller. All list elements that have indexes greater than *iIndex* at the time of the call will have their index in the list incremented by one. Permitted values for *iIndex* are 0 through *n*, where *n* is the number of elements in the list at the time of the call. A value of *n* appends a new value to the end of the list.

4.2.14.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.14.1.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.14.2 IdmaEditListOfId::ReplacId

This method must be supported in every implementation of this interface.

4.2.14.2.1 Syntax

```
DmaRC IdmaEditListOfId::ReplacId (
    DmaIndex32 iIndex,
    pDmaId pldValue)
```

4.2.14.2.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	The ordinal position in the list where the value will be replaced.
<i>pldValue</i>	input	The new Id value that will replace the value in the list.

4.2.14.2.3 Description

Replaces a DMA identifier value in a list with the supplied Id value given an ordinal position value as input. Valid values for *ilIndex* are 0 through $n-1$, where n is the number of elements in the list at the time of the call.

4.2.14.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.14.2.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.15 IdmaEditListOfInteger32

This interface provides methods for modifying lists that contain 32 bit integer values.

This interface offers all the methods of the *IdmaEditList* interface, as well as those listed below.

4.2.15.1 IdmaEditListOfInteger32::InsertInteger32

This method must be supported in every implementation of this interface.

4.2.15.1.1 Syntax

DmaRC IdmaEditListOfInteger32::InsertInteger32 (
DmaIndex32 ilIndex,
DmaInteger32 lIntegerValue)

4.2.15.1.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>lIntegerValue</i>	input	The integer value that will be inserted into the list.

4.2.15.1.3 Description

This function inserts an integer value into the list given on an ordinal position provided by the caller. All list elements that have indexes greater than *ilIndex* at the time of the call will have their index in the list incremented by one. Permitted values for *ilIndex* are 0 through *n*, where *n* is the number of elements in the list at the time of the call. A value of *n* appends a new value to the end of the list.

4.2.15.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.15.1.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.15.2 IdmaEditListOfInteger32::ReplaceInteger32

This method must be supported in every implementation of this interface.

4.2.15.2.1 Syntax

DmaRC IdmaEditListOfInteger32::ReplaceInteger32 (
DmaIndex32 iIndex,
DmaInteger32 lIntegerValue)

4.2.15.2.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>lIntegerValue</i>	input	The new integer value that will replace the value in the list.

4.2.15.2.3 Description

Replaces a integer value in a list with the supplied integer value given an ordinal position value as input. Valid values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

4.2.15.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.15.2.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.16 IdmaEditListOfObject

This interface provides the methods for modifying lists that contain references to objects. (This interface offers all the methods of the *IdmaEditList* interface, as well as those listed below.)

4.2.16.1 IdmaEditListOfObject::InsertObject

This method must be supported in every implementation of this interface.

4.2.16.1.1 Syntax

```
DmaRC IdmaEditListOfObject::InsertObject (
    DmaIndex32 ilIndex,
    Dmapv pObjectValue)
```

4.2.16.1.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>pObjectValue</i>	input	The object that will be inserted into the list.

4.2.16.1.3 Description

This function inserts an object into a list based on an ordinal position provided by the caller. All list elements that have indexes greater than *ilIndex* at the time of the call will have their index in the list incremented by one. Permitted values for *ilIndex* are 0 through *n*, where *n* is the number of elements in the list at the time of the call. A value of *n* appends a new value to the end of the list.

The object is inserted into the list by reference. Changes to the object outside the context of the list will result in side-effects on the list.

4.2.16.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.16.1.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.16.2 IdmaEditListOfObject::ReplaceObject

This method must be supported in every implementation of this interface.

4.2.16.2.1 Syntax

```
DmaRC IdmaEditListOfObject::ReplaceObject (
    DmaIndex32 ilIndex,
    Dmapv pLObjectValue)
```

4.2.16.2.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>pLObjectValue</i>	input	The new object that will replace the object in the list.

4.2.16.2.3 Description

Replaces an object value in a list with the supplied object value given an ordinal position value as input. Valid values for *ilIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

The object found in the list will be released using the method IUnknown::Release. The new object is inserted into the list by reference. Changes to the object outside the context of the list will result in side-effects on the list.

4.2.16.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.16.2.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.17 IdmaEditListOfString

This interface provides methods for modification of lists that contain strings of character data.

This interface offers all the methods of the IdmaEditList interface, as well as those listed below.

4.2.17.1 IdmaEditListOfString::InsertString

This method must be supported in every implementation of this interface.

4.2.17.1.1 Syntax

```
DmaRC IdmaEditListOfString::InsertString (
    DmaIndex32 iIndex,
    pDmaString pStringValue)
```

4.2.17.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>pStringValue</i>	input	The string value that will be inserted into the list.

4.2.17.1.3 Description

This function inserts a string value into the list given on an ordinal position provided by the caller. All list elements that have indexes greater than *iIndex* at the time of the call will have their index in the list incremented by one. Permitted values for *iIndex* are 0 through *n*, where *n* is the number of elements in the list at the time of the call. A value of *n* appends a new value to the end of the list.

4.2.17.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.17.1.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.17.2 IdmaEditListOfString::ReplaceString

This method must be supported in every implementation of this interface.

4.2.17.2.1 Syntax

```
DmaRC IdmaEditListOfString::ReplaceString (
    DmaIndex32 iIndex,
    pDmaString pStringValue)
```

4.2.17.2.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list where the value will be placed.
<i>pStringValue</i>	input	The new string value that will replace the value in the list.

4.2.17.2.3 Description

Replaces a string value in a list with the supplied string value given an ordinal position value as input. Valid values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

4.2.17.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.17.2.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.

4.2.18 IdmaEditProperties

This interface allows writing and deleting DMA property values of any DMA base property type: Binary, Boolean, DateTime, Float, Identifier, Integer, Object, or String. Properties may be chosen either by their index in the object's class description list, or by their property identifier (e.g. dmaProp_Rendition-sPresent). Both indices and property ids can be found from the object's meta-data. Some Property Ids are well known and can be used directly without investigation of the meta-data.

The methods used to access DMA Properties are segmented into two interfaces, the methods that allow readonly access to the properties (IdmaProperties), and the methods that allow modification of the properties (IdmaEditProperties). The IdmaEditProperties interface is available only on objects that are editable. It offers all the methods of the IdmaProperties interface, as well as those listed below.

When any method in this interface is called successfully, the UpdatePending property for the target object is set to TRUE unconditionally, even if the new value that the property is being set to is exactly the same as the old value.

4.2.18.1 IdmaEditProperties::DeletePropValById

Support for this method is optional.

4.2.18.1.1 Syntax

DmaRC **IdmaEditProperties::DeletePropValById** (
pDmald *pPropertyId*)

4.2.18.1.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be obtained. This identifier may be well-known or obtained from the meta-data.

4.2.18.1.3 Description

Deletes the property value for the property selected by its identifier, which is either "well-known" or obtained from the meta-data. The effect is that a subsequent call to a GetPropVal method will return DMARC_VALUE_NOT_SET.

Any value set for the property will be appropriately freed as part of this operation.

This operation must fail, with return code DMARC_ILLEGAL_OPERATION, if applied to a list or enumeration property.

4.2.18.1.4 Return Values

Name	Description
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_ILLEGAL_OPERATION	The operation is disallowed by the DMA Specification.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.18.1.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.2 IdmaEditProperties::DeletePropValByIndex

Support for this method is optional.

4.2.18.2.1 Syntax

DmaRC **IdmaEditProperties::DeletePropValByIndex** (
DmaIndex32 *iIndex*)

4.2.18.2.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be modified. This index may be obtained from the meta-data.

4.2.18.2.3 Description

Deletes the property value for the property identified by the index, which is defined by the object's Class Description. The effect is that a subsequent call to a GetPropVal method will return DMARC_VALUE_NOT_SET. Any value set for the property will be appropriately freed as part of this operation.

This operation must fail, with return code DMARC_ILLEGAL_OPERATION, if applied to a list or enumeration property.

4.2.18.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_ILLEGAL_OPERATION	The operation is disallowed by the DMA Specification.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.18.2.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.3 IdmaEditProperties::PutPropValBinaryByld

Support for this method is optional.

4.2.18.3.1 Syntax

DmaRC **IdmaEditProperties::PutPropValBinaryByld** (
 pDmald *pPropertyId*,
 DmaBinaryValue **pBinaryValue*)

4.2.18.3.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be modified. This identifier may be well-known or obtained from the meta-data.
<i>pBinaryValue</i>	input	A pointer to the buffer containing the binary value.

4.2.18.3.3 Description

Assigns a Property Value as a fixed number of bytes. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.18.3.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.3.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.4 IdmaEditProperties::PutPropValBinaryByIndex

Support for this method is optional.

4.2.18.4.1 Syntax

DmaRC **IdmaEditProperties::PutPropValBinaryByIndex** (
DmaIndex32 *iIndex*,
DmaBinaryValue **pBinaryValue*)

4.2.18.4.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be modified. This index may be obtained from the meta-data.
<i>pBinaryValue</i>	input	A pointer to the buffer containing the binary value.

4.2.18.4.3 Description

Assigns a Property Value as a fixed number of bytes. This is one of the basic data types used for property values. The property of interest is located using its index, which is defined by the object's Class Description.

4.2.18.4.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.4.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.5 IdmaEditProperties::PutPropValBooleanById

Support for this method is optional.

4.2.18.5.1 Syntax

DmaRC **IdmaEditProperties::PutPropValBooleanById**
 (pDmald *pPropertyId*,
 DmaBoolean *bBooleanValue*)

4.2.18.5.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be modified. This identifier may be well-known or obtained from the meta-data.
<i>bBooleanValue</i>	input	The boolean value of the Property to be assigned.

4.2.18.5.3 Description

Assigns a Property Value as a boolean. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.18.5.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.5.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.6 IdmaEditProperties::PutPropValBooleanByIndex

Support for this method is optional.

4.2.18.6.1 Syntax

DmaRC **IdmaEditProperties::PutPropValBooleanByIndex** (
DmaIndex32 *iIndex*,
DmaBoolean *bBooleanValue*)

4.2.18.6.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be modified. This index may be obtained from the meta-data.
<i>bBooleanValue</i>	input	The boolean value of the Property to be assigned.

4.2.18.6.3 Description

Assigns a Property Value as a boolean. This is one of the basic data types used for property values. The property of interest is located using its index, which is defined by the object's meta-data.

4.2.18.6.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.6.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.7 IdmaEditProperties::PutPropValDateTimeById

Support for this method is optional.

4.2.18.7.1 Syntax

DmaRC **IdmaEditProperties::PutPropValDateTimeById** (
 pDmald *pPropertyId*,
 pDmaDateTime *pDateTimeValue*)

4.2.18.7.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be modified. This identifier may be well-known or obtained from the meta-data.
<i>pDateTimeValue</i>	input	A pointer to the DateTime to be assigned to the property.

4.2.18.7.3 Description

Assigns a DateTime valued property. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.18.7.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.7.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.8 IdmaEditProperties::PutPropValDateTimeByIndex

Support for this method is optional.

4.2.18.8.1 Syntax

DmaRC **IdmaEditProperties::PutPropValDateTimeByIndex** (
DmaIndex32 *iIndex*,
pDmaDateTime *pDateTimeValue*)

4.2.18.8.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be modified. This index may be obtained from the meta-data.
<i>pDateTimeValue</i>	input	A pointer to the DateTime to be assigned to the property.

4.2.18.8.3 Description

Assigns a DateTime valued property. The property of interest is indicated by its index in the set of properties for the object.

4.2.18.8.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.8.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.9 IdmaEditProperties::PutPropValFloat64ById

Support for this method is optional.

4.2.18.9.1 Syntax

DmaRC **IdmaEditProperties::PutPropValFloat64ById** (
 pDmald *pPropertyId*,
 DmaFloat64 *fFloatValue*)

4.2.18.9.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be modified. This identifier may be well-known or obtained from the meta-data.
<i>fFloatValue</i>	input	The float value of the Property to be assigned.

4.2.18.9.3 Description

Assigns a Property Value as a float, which must be 64 bits and signed. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.18.9.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.9.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.10 IdmaEditProperties::PutPropValFloat64ByIndex

Support for this method is optional.

4.2.18.10.1 Syntax

DmaRC **IdmaEditProperties::PutPropValFloat64ByIndex** (
DmaIndex32 *iIndex*,
DmaFloat64 *fFloatValue*)

4.2.18.10.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be modified. This index may be obtained from the meta-data.
<i>fFloatValue</i>	input	The float value of the Property to be assigned.

4.2.18.10.3 Description

Assigns a Property Value as a float, which must be 64 bits and signed. This is one of the basic data types used for property values. The property of interest is located using its index, which is defined by the object's meta-data.

4.2.18.10.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.10.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.11 IdmaEditProperties::PutPropValIdById

Support for this method is optional.

4.2.18.11.1 Syntax

DmaRC **IdmaEditProperties::PutPropValIdById** (
 pDmald *pPropertyId*,
 pDmald *pIdValue*)

4.2.18.11.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be modified. This identifier may be well-known or obtained from the meta-data.
<i>pIdValue</i>	input	A pointer to the Id to be assigned to the property.

4.2.18.11.3 Description

Assigns a DMA identifier valued property. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.18.11.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.11.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.12 IdmaEditProperties::PutPropValldByIndex

Support for this method is optional.

4.2.18.12.1 Syntax

DmaRC **IdmaEditProperties::PutPropValldByIndex** (
DmaIndex32 *ilIndex*,
pDmaId *pldValue*)

4.2.18.12.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	An index for the property that is to be modified. This index may be obtained from the meta-data.
<i>pldValue</i>	input	A pointer to the Id to be assigned to the property.

4.2.18.12.3 Description

Assigns a DMA identifier valued property. The property of interest is indicated by its index in the set of properties for the object.

4.2.18.12.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.12.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.13 IdmaEditProperties::PutPropValInteger32ById

Support for this method is optional.

4.2.18.13.1 Syntax

DmaRC **IdmaEditProperties::PutPropValInteger32ById** (
 pDmald *pPropertyId*,
 DmaInteger32 *IntegerValue*)

4.2.18.13.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be modified. This identifier may be well-known or obtained from the meta-data.
<i>IntegerValue</i>	input	The integer value of the Property to be assigned.

4.2.18.13.3 Description

Assigns a Property Value as a long integer, which must be 32 bits and signed. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.18.13.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.13.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.14 IdmaEditProperties::PutPropValInteger32ByIndex

Support for this method is optional.

4.2.18.14.1 Syntax

DmaRC **IdmaEditProperties::PutPropValInteger32ByIndex** (
DmaIndex32 *iIndex*,
DmaInteger32 *IntegerValue*)

4.2.18.14.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be modified. This index may be obtained from the meta-data.
<i>IntegerValue</i>	input	The integer value of the Property to be assigned.

4.2.18.14.3 Description

Assigns a Property Value as a long integer, which must be 32 bits and signed. This is one of the basic data types used for property values. The property of interest is located using its index, which is defined by the object's meta-data.

4.2.18.14.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.14.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.15 IdmaEditProperties::PutPropValObjectById

Support for this method is optional.

4.2.18.15.1 Syntax

DmaRC **IdmaEditProperties::PutPropValObjectById** (
 pDmald *pPropertyId*,
 Dmapv *pObjectValue*)

4.2.18.15.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be assigned. This identifier may be well-known or obtained from the meta-data.
<i>pObjectValue</i>	input	A interface for the DMA object to be assigned to the property.

4.2.18.15.3 Description

Assigns a Property Value, in object form, to a property. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data. This can be used, for instance, to assign a property value list into its owning object.

This operation must fail, with return code DMARC_ILLEGAL_OPERATION, if applied to a list or enumeration property.

4.2.18.15.4 Return Values

Name	Description
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_ILLEGAL_OPERATION	The operation is disallowed by the DMA Specification.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.15.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.16 IdmaEditProperties::PutPropValObjectByIndex

Support for this method is optional.

4.2.18.16.1 Syntax

DmaRC **IdmaEditProperties::PutPropValObjectByIndex** (
DmaIndex32 *iIndex*,
Dmapv *pObjectValue*)

4.2.18.16.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be modified. This index may be obtained from the meta-data.
<i>pObjectValue</i>	input	A interface for the DMA object to be assigned to the property.

4.2.18.16.3 Description

Assigns a Property Value, in object form, to a property. The property of interest is located using its index, which is defined by the object's meta-data. This can be used, for instance, to assign a property value list into its owning object.

This operation must fail, with return code DMARC_ILLEGAL_OPERATION, if applied to a list or enumeration property.

4.2.18.16.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_ILLEGAL_OPERATION	The operation is disallowed by the DMA Specification.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.16.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.17 IdmaEditProperties::PutPropValStringById

Support for this method is optional.

4.2.18.17.1 Syntax

DmaRC **IdmaEditProperties::PutPropValStringById** (
 pDmaId *pPropertyId*,
 pDmaString *pStringValue*)

4.2.18.17.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be modified. This identifier may be well-known or obtained from the meta-data.
<i>pStringValue</i>	input	A pointer to the Property Value string to be assigned to the property.

4.2.18.17.3 Description

Assigns a string valued property. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.18.17.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.17.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.18.18 IdmaEditProperties::PutPropValStringByIndex

Support for this method is optional.

4.2.18.18.1 Syntax

DmaRC **IdmaEditProperties::PutPropValStringByIndex** (
DmaIndex32 *iIndex*,
pDmaString *pStringValue*)

4.2.18.18.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be modified. This index may be obtained from the meta-data.
<i>pStringValue</i>	input	A pointer to the Property Value string to be assigned to the property.

4.2.18.18.3 Description

Assigns a string valued property. The property of interest is indicated by its index in the set of properties for the object.

4.2.18.18.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.

4.2.18.18.5 Deferred Return Values

Name	Description
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.

4.2.19 IdmaEnumOfObject

This interface provides the methods for handling enumerators that contain references to DMA Objects.

4.2.19.1 IdmaEnumOfObject::GetNextObject

This method must be supported in every implementation of this interface.

4.2.19.1.1 Syntax

```
DmaRC IdmaEnumOfObject::GetNextObject (
    DmaIndex32 iNumItems,
    IUnknown **pArray,
    pDmaIndex32 piNumReturned)
```

4.2.19.1.2 Parameters

Name	Mode	Description
<i>iNumItems</i>	input	Indicates the number of items requested from the enumeration object.
<i>pArray</i>	output	An array with at least <i>uiNumItems</i> elements. The array elements will be set to pointers to the IUnknown interface for the objects returned.
<i>piNumReturned</i>	output	Returns the actual number of objects returned in <i>pArray</i> .

4.2.19.1.3 Description

This function will return the next *iNumItems* objects in the object enumeration.

If there are N elements remaining, but N is less than *iNumItems* and greater than zero, then the first N elements of the array will be valid and the method will return DMARC_OK. The actual number will be returned in *piNumReturned*.

If there are no items to return, **piNumReturned* will be set to zero and an error code will be returned.

If an error occurs while generating the first object, an error code will be returned, no objects will be returned, and **piNumReturned* will be set to zero. The enumeration will be positioned at the object in error.

If an error occurs while generating an object other than the first object, as many objects as were successfully enumerated will be returned, **piNumReturned* will be set to the number of objects successfully returned, and DMARC_OK will be returned. The enumeration will be positioned on the object that caused the error.

If an error is returned, and *GetNextObject* is called again, an error may or may not be returned, depending upon whether the error condition still exists. If the error condition no longer exists (e.g., a "low memory" condition has been resolved), the enumeration of objects will proceed normally starting at the current position of the enumeration.

Once all objects have been enumerated, DMARC_AT_ENUM_END is returned on all subsequent calls

The array `pArray` will contain references to the `IUnknown` interface for each of the returned objects. These interfaces must be released using the `IUnknown::Release` method when no longer needed by the client, and in all circumstances **piNumReturned* reflects exactly the number of such interfaces. However, the client application may rely upon the fact that this number will be non-zero only in the case that the method returns `DMARC_OK`.

4.2.19.1.4 Return Values

Name	Description
DMARC_AT_ENUM_END	The enumerator object is positioned at the end of the component objects.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.

4.2.20 IdmaList

Lists are a fundamental feature in managing documents, so there are several operations available for manipulating them and the objects contained within them. Items within a list are kept in inserted order and can be identified by an index (ordinal position).

The methods used to access a DMA List are segmented into two groups of interfaces, the methods that allow readonly access to the list (`IdmaList`), and the methods that allow modification of the list (`IdmaEditList`). The `IdmaEditList` interface is only available on list objects that are editable. For each list datatype, there is a specific interface that offers methods particular to that datatype, as well as all the methods of `IdmaList`, (e.g., `IdmaListOfString`).

4.2.20.1 IdmaList::GetElementCount

This method must be supported in every implementation of this interface.

4.2.20.1.1 Syntax

DmaRC IdmaList::GetElementCount (
pDmaIndex32 piCount)

4.2.20.1.2 Parameters

Name	Mode	Description
<i>piCount</i>	output	A pointer to the Count location.

4.2.20.1.3 Description

Determines the number of elements in a list object, which can be used to iterate through the list for processing. The number returned is only valid at the time for inquiry, since any inserts or deletes will change the underlying list and render the count inaccurate.

4.2.20.1.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.21 IdmaListOfBinary

This interface provides the methods for handling lists that contain buffers of binary data.

This interface offers all the methods of the IdmaList interface, as well as those listed below.

4.2.21.1 IdmaListOfBinary::GetBinary

This method must be supported in every implementation of this interface.

4.2.21.1.1 Syntax

DmaRC **IdmaListOfBinary::GetBinary** (
DmaIndex32 *ilIndex*,
DmaBinaryValue **pBinaryValue*)

4.2.21.1.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	The ordinal position in the list to retrieve the binary value from.
<i>pBinaryValue</i>	output	A pointer to the returned binary value.

4.2.21.1.3 Description

This function will return the binary value that resides in the *ilIndex*-th position in the list.

Valid values for *ilIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

The caller is responsible for allocating and freeing the structure pointed to by the *pBinaryValue* parameter. The memory pointed to by the *pBinaryValue*-

>pbBytes field is allocated by the callee and must be freed by the caller. This field need not be pre-initialised to NULL by the caller, and must be set to NULL by the callee if any return code other than DMARC_OK is returned.

4.2.21.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.22 IdmaListOfBoolean

This interface provides the methods for handling lists that contain boolean values.

This interface offers all the methods of the IdmaList interface, as well as those listed below.

4.2.22.1 IdmaListOfBoolean::GetBoolean

This method must be supported in every implementation of this interface.

4.2.22.1.1 Syntax

```
DmaRC IdmaListOfBoolean::GetBoolean (
    DmaIndex32 ilIndex,
    pDmaBoolean pbBooleanValue)
```

4.2.22.1.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	The ordinal position in the list to retrieve the boolean value from.
<i>pbBooleanValue</i>	output	Returns the boolean value at <i>ilIndex</i> in the list.

4.2.22.1.3 Description

This function will return the boolean value that resides in the *ilIndex*-th position in the list.

Valid values for *ilIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

4.2.22.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.23 IdmaListOfDateTime

This interface provides methods for handling lists that contain dates.

This interface offers all the methods of the IdmaList interface, offers all the methods of as well as those listed below.

4.2.23.1 IdmaListOfDateTime::GetDateTime

This method must be supported in every implementation of this interface.

4.2.23.1.1 Syntax

```
DmaRC IdmaListOfDateTime::GetDateTime (
    DmaIndex32 iIndex,
    ppDmaDateTime ppValue)
```

4.2.23.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list of the value of interest.
<i>ppValue</i>	output	Returns the value at <i>iIndex</i> in the list.

4.2.23.1.3 Description

This function will return the value that resides in the *iIndex*-th position in the list.

Valid values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

4.2.23.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.24 IdmaListOfFloat64

This interface provides the methods for handling lists that contain 64 bit float values.

This interface offers all the methods of the IdmaList interface, as well as those listed below.

4.2.24.1 IdmaListOfFloat64::GetFloat64

This method must be supported in every implementation of this interface.

4.2.24.1.1 Syntax

DmaRC **IdmaListOfFloat64::GetFloat64** (
DmaIndex32 *iIndex*,
pDmaFloat64 *pfFloatValue*)

4.2.24.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list to retrieve the float value from.
<i>pfFloatValue</i>	output	Returns the float value at <i>iIndex</i> in the list.

4.2.24.1.3 Description

This function will return the float value that resides in the *iIndex*-th position in the list.

Valid values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

4.2.24.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.25 IdmaListOfId

This interface provides methods for handling lists that contain DMA identifiers

This interface offers all the methods of the IdmaList interface, as well as those listed below.

4.2.25.1 IdmaListOfId::GetId

This method must be supported in every implementation of this interface.

4.2.25.1.1 Syntax

```
DmaRC IdmaListOfId::GetId (
    DmaIndex32 iIndex,
    pDmaId pValue)
```

4.2.25.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list of the value of interest.
<i>pValue</i>	output	Pointer to client allocated storage which the method will populate with the value at <i>iIndex</i> in the list.

4.2.25.1.3 Description

This function will return the value that resides in the *iIndex*-th position in the list.

Valid values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

4.2.25.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.26 IdmaListOfInteger32

This interface provides the methods for handling lists that contain 32 bit integer values.

This interface offers all the methods of the `IdmaList` interface, as well as those listed below.

4.2.26.1 `IdmaListOfInteger32::GetInteger32`

This method must be supported in every implementation of this interface.

4.2.26.1.1 Syntax

```
DmaRC IdmaListOfInteger32::GetInteger32 (  
    DmaIndex32 iIndex,  
    pDmaInteger32 plIntegerValue)
```

4.2.26.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list to retrieve the integer value from.
<i>plIntegerValue</i>	output	Returns the integer value at <i>iIndex</i> in the list.

4.2.26.1.3 Description

This function will return the integer value that resides in the *iIndex*-th position in the list.

Valid values for *iIndex* are 0 through $n-1$, where n is the number of elements in the list at the time of the call.

4.2.26.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.27 `IdmaListOfObject`

This interface provides the methods for handling lists that contain references to objects.

This interface offers all the methods of the `IdmaList` interface, as well as those listed below.

4.2.27.1 `IdmaListOfObject::GetObject`

This method must be supported in every implementation of this interface.

4.2.27.1.1 Syntax

```
DmaRC IdmaListOfObject::GetObject (
    DmaIndex32 iIndex,
    DMA_REFIID riid,
    pDmapv ppIObjectValue)
```

4.2.27.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position in the list to retrieve the object from.
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppIObjectValue</i>	output	Returns a pointer to the interface of the returned object, which is set to NULL for unsuccessful return codes.

4.2.27.1.3 Description

This function will return the interface for the object that resides in the *iIndex*-th position in the list.

Valid values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

If the returned object does not support the interface specified by *riid*, the operation will fail and the result code DMARC_BAD_INTERFACE will be returned. In case the operation is successful, the returned interface must be released using the method IUnknown::Release.

4.2.27.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OBJECT_DELETED	The object has been deleted since it was retrieved.
DMARC_OK	(S_OK) Success.
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.

4.2.28 IdmaListOfString

This interface provides methods for handling lists that contain strings of character data.

This interface offers all the methods of the IdmaList interface, as well as those listed below.

4.2.28.1 IdmaListOfString::GetString

This method must be supported in every implementation of this interface.

4.2.28.1.1 Syntax

```
DmaRC IdmaListOfString::GetString (
    DmaIndex32 iIndex,
    ppDmaString ppStringValue)
```

4.2.28.1.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	The ordinal position of the value of interest.
<i>ppStringValue</i>	output	Returns the string value at <i>iIndex</i> in the list.

4.2.28.1.3 Description

This function will return the string value that resides in the *iIndex*-th position in the list.

Valid values for *iIndex* are 0 through *n*-1, where *n* is the number of elements in the list at the time of the call.

4.2.28.1.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.29 IdmaObject

This is a base interface which must be supported by all DMA objects.

4.2.29.1 IdmaObject::IsOfClass

This method must be supported in every implementation of this interface.

4.2.29.1.1 Syntax

DmaRC **IdmaObject::IsOfClass** (
 pDmald *pClassId*,
 pDmaBoolean *pbIsOfClass*)

4.2.29.1.2 Parameters

Name	Mode	Description
<i>pClassId</i>	input	A pointer to the identifier of the class.
<i>pbIsOfClass</i>	output	Returns DMA_TRUE if the object conforms to the selected Class Id.

4.2.29.1.3 Description

Returns DMA_TRUE if and only if the object conforms to the selected class. To be conformant, the specified class ID must be an identifier for a class for which the object's class is a subclass. All DMA objects should return DMA_TRUE if the specified class ID is dmaClass_DMA.

4.2.29.1.4 Return Values

Name	Description
DMARC_BAD_CLASSID	The supplied identifier does not reference an available class of object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.30 IdmaObjectFactory

This interface provides access to object creation services of a DMA System object, DocSpace object or Scope object.

4.2.30.1 IdmaObjectFactory::CreateObject

This method must be supported in every implementation of this interface.

4.2.30.1.1 Syntax

DmaRC **IdmaObjectFactory::CreateObject** (
 pDmald *pClassId*,
 DMA_REFIID *riid*,
 pDmapv *ppIObject*)

4.2.30.1.2 Parameters

Name	Mode	Description
<i>pClassId</i>	input	A pointer to the identifier of the class of object to be created. This identifier may be well-known or obtained from the meta-data.
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppIObject</i>	output	Returns a pointer for the requested interface for the object created, which will be NULL if unsuccessful.

4.2.30.1.3 Description

This will create a new object of the class indicated by *pClassId* in the client's process space. The object's properties will be set with default values, and may be changed using the *IdmaEditProperties* interface.

If the returned object does not support the interface specified by *riid*, the operation will fail and the result code *DMARC_BAD_INTERFACE* will be returned. In case the operation is successful, the returned interface must be released using the method *IUnknown::Release*.

The method gives gives *DMARC_NOT_CREATABLE* errors for objects which are inappropriate to generate from the object supporting the method. For example, a System object will not support creating independently persistable objects.

4.2.30.1.4 Return Values

Name	Description
<i>DMARC_BAD_CLASSID</i>	The supplied identifier does not reference an available class of object.
<i>DMARC_BAD_INTERFACE</i>	(<i>E_NOINTERFACE</i>) The requested interface is not supported by this object.
<i>DMARC_BAD_PARAMETER</i>	(<i>E_INVALIDARG</i>) Invalid input parameter.
<i>DMARC_NOT_CREATABLE</i>	Creation of objects of the specified class is not supported.
<i>DMARC_OK</i>	(<i>S_OK</i>) Success.

4.2.31 IdmaOIID

Methods that deal specifically with DMA object instance identifiers.

4.2.31.1 IdmaOIID::GetDocSpaceId

This method must be supported in every implementation of this interface.

4.2.31.1.1 Syntax

DmaRC **IdmaOIID::GetDocSpaceId** (
pDmaString *pOIID*,
pDmald *pDocSpaceId*)

4.2.31.1.2 Parameters

Name	Mode	Description
<i>pOIID</i>	input	Pointer to the OIID
<i>pDocSpaceId</i>	output	Pointer to a caller-allocated Dmald into which the docspace-id component of the OIID input parameter is extracted.

4.2.31.1.3 Description

This method extracts the docspace-id from the OIID input parameter and returns the docspace-id in a Dmald structure.

There is a small well defined set of DMA return codes returned from this method. There is no DMA supplied mechanism for mapping these return codes into a meaningful message (e.g., IdmaSystem::GetResultCodeDescription), until a DMA System object becomes available.

4.2.31.1.4 Return Values

Name	Description
DMARC_BAD_OIID	The supplied buffer does not contain a valid OIID.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.31.2 IdmaOIID::GetGuid

This method must be supported in every implementation of this interface.

4.2.31.2.1 Syntax

DmaRC **IdmaOIID::GetGuid** (
pDmaString *pOIID*,
pDmald *pObjectGUID*)

4.2.31.2.2 Parameters

Name	Mode	Description
<i>pOIID</i>	input	Pointer to a DMA OIID
<i>pObjectGUID</i>	output	Pointer to a Dmald into which the object-guid component of the OIID input parameter is extracted

4.2.31.2.3 Description

This method extracts the optional object-guid from the OIID input parameter and returns the object-guid in a Dmald structure.

There is a small well defined set of DMA return codes returned from this method. There is no DMA supplied mechanism for mapping DMA return codes into a meaningful message (e.g., `IdmaSystem::GetResultCodeDescription`), until a DMA System object becomes available.

4.2.31.2.4 Return Values

Name	Description
DMARC_BAD_OIID	The supplied buffer does not contain a valid OIID.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_FOUND	Requested item not found.
DMARC_OK	(S_OK) Success.

4.2.31.3 IdmaOIID::GetObjectIdText

Support for this method is optional.

4.2.31.3.1 Syntax

```
DmaRC IdmaOIID::GetObjectIdText (
    pDmaString pOIID,
    ppDmaString ppObjectText)
```

4.2.31.3.2 Parameters

Name	Mode	Description
<i>pOIID</i>	input	The OIID to be parsed.
<i>ppObjectText</i>	output	The returned object Id component

4.2.31.3.3 Description

This will extract the object identifier component from an OIID, if it is possible to successfully parse and locate this component.

4.2.31.3.4 Return Values

Name	Description
DMARC_BAD_OIID	The supplied buffer does not contain a valid OIID.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_NOT_FOUND	Requested item not found.
DMARC_OK	(S_OK) Success.

4.2.31.4 IdmaOIID::GetSystemId

This method must be supported in every implementation of this interface.

4.2.31.4.1 Syntax

```
DmaRC IdmaOIID::GetSystemId (
    pDmaString pOIID,
    pDmald pSystemId)
```

4.2.31.4.2 Parameters

Name	Mode	Description
<i>pOIID</i>	input	Pointer to the OIID
<i>pSystemId</i>	output	Pointer to a Dmald into which the system-id component of the OIID input parameter is extracted

4.2.31.4.3 Description

This method extracts the system-id from the OIID input parameter and returns the system-id in a Dmald structure.

There is a small well defined set of DMA return codes returned from this method. There is no DMA supplied mechanism for mapping these return codes into a meaningful message (e.g., IdmaSystem::GetResultCodeDescription), until a DMA System object becomes available.

4.2.31.4.4 Return Values

Name	Description
DMARC_BAD_OIID	The supplied buffer does not contain a valid OIID.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.32 IdmaProgressCallback

Methods supporting the ability of DMA methods to report progress to their callers. This interface may be implemented by a client-provided COM object; support is not required by any of the DMA objects.

4.2.32.1 IdmaProgressCallback::ReportProgress

This method must be supported in every implementation of this interface.

4.2.32.1.1 Syntax

DmaRC **IdmaProgressCallback::ReportProgress** (
DmaRC *rcProgressStatus*,
DmaUInteger32 *ulProgressNumerator*,
DmaUInteger32 *ulProgressDenominator*,
DmaBoolean *bAbortAllowed*)

4.2.32.1.2 Parameters

Name	Mode	Description
<i>rcProgressStatus</i>	input	A DMA return code that indicates the status of the underlying operation's progress at this point. DMARC_OK always indicates that the operation is proceeding normally. Any other value is for a defined exception that the requester has the option of ignoring.
<i>ulProgressNumerator</i>	input	The numerator of a fraction that represents fraction of completion of the underlying operation.
<i>ulProgressDenominator</i>	input	The denominator of a fraction that represents fraction of completion. If the <i>ulProgressDenominator</i> is nonzero, then $\text{ulProgressNumerator} / \text{ulProgressDenominator} * 100 \%$ is the percentage of completion of the operation. If <i>ulProgressDenominator</i> is zero, the caller is unable to supply any information about the degree of completeness of the operation.
<i>bAbortAllowed</i>	input	If DMA_TRUE, the underlying operation in progress can be aborted at this point; if DMA_FALSE it cannot (and the return code from ReportProgress will be ignored). Successive calls can pass in different values for <i>bAbortAllowed</i> .

4.2.32.1.3 Description

This method provides a way for potentially long-running DMA methods to report back on their progress. When a client supplies an IdmaProgressCallback

interface pointer to a method which accepts such an argument, that method may provide a 'percentage done' indication via ReportProgress(). ReportProgress() also allows the caller to request that the method may be aborted.

Implementations of ReportProgress() must always return to their caller, and (because their execution delays the underlying method) should execute quickly. They should normally return DMARC_OK. Any other error code will be considered as a request to abort the underlying method.

There are no guarantees that ReportProgress() will ever be called, that a requested abort will take place, or that a non-zero value will ever be supplied for nProgressDenominator. If nProgressDenominator is nonzero, the callback procedure cannot rely on the percentage done being non-decreasing.

There are certain guarantees. (1) Once the denominator is nonzero, it stays nonzero. (2) If the denominator is ever nonzero, the percentage done will never be more than 100%, and the percent done can increase, decrease, or stay the same. (3) The client must be written so that the ReportProgress method might never be called, and, if it is, it may be called repeatedly, even after it instructs the underlying method to abort.

4.2.32.1.4 Return Values

Name	Description
DMARC_ABORT	The progress callback indicated that the method should be aborted.
DMARC_OK	(S_OK) Success.

4.2.33 IdmaProperties

This interface provides read-only access to DMA property values of any DMA base property type: Binary, Boolean, DateTime, Float, Identifier, Integer, Object, or String. Properties may be chosen either by their index in the object's class description list, or by their property identifier (e.g. dmaProp_Rendition-sPresent). Both indices and property ids can be found from the object's meta-data. Some Property Ids are well known and can be used directly without investigation of the meta-data.

The methods used to access DMA Properties are segmented into two interfaces, the methods that allow readonly access to the properties (IdmaProperties), and the methods that allow modification of the properties (IdmaEditProperties).

When a caller requests the value of a property of type String, DateTime, or Binary, the called method allocates storage for the property value (using the DMA system allocator), and passes back a pointer to the allocated storage. It is the caller's responsibility to free that storage when it is no longer needed -- again using the DMA system allocator.

4.2.33.1 IdmaProperties::GetPropValBinaryById

Support for this method is optional.

4.2.33.1.1 Syntax

DmaRC IdmaProperties::GetPropValBinaryById (
pDmald *pPropertyId*,
DmaBinaryValue **pBinaryValue*)

4.2.33.1.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be obtained. This identifier may be well-known or obtained from the meta-data.
<i>pBinaryValue</i>	output	A pointer to the DmaBinaryValue structure that describes the returned value.

4.2.33.1.3 Description

Returns a binary property value -- a fixed number of bytes of arbitrary format. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

The caller is responsible for allocating and freeing the structure pointed to by the pBinaryValue parameter. The memory pointed to by the pBinaryValue->pbBytes field is allocated by the callee and must be freed by the caller. This field need not be pre-initialised to NULL by the caller, and must be set to NULL by the callee if any return code other than DMARC_OK is returned.

4.2.33.1.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.2 IdmaProperties::GetPropValBinaryByIndex

Support for this method is optional.

4.2.33.2.1 Syntax

DmaRC IdmaProperties::GetPropValBinaryByIndex (
DmaIndex32 *iIndex*,
DmaBinaryValue **pBinaryValue*)

4.2.33.2.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be obtained. This index may be obtained from the meta-data.
<i>pBinaryValue</i>	output	A pointer to the Property Value location. If unsuccessful, the bytes at this location will remain unchanged.

4.2.33.2.3 Description

Returns a binary property value -- a fixed number of bytes of arbitrary format. This is one of the basic data types used for property values. The property of interest is located using its index, which is defined by the object's Class Description.

The caller is responsible for allocating and freeing the structure pointed to by the *pBinaryValue* parameter. The memory pointed to by the *pBinaryValue->pbBytes* field is allocated by the callee and must be freed by the caller. This field need not be pre-initialised to NULL by the caller, and must be set to NULL by the callee if any return code other than DMARC_OK is returned.

4.2.33.2.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.3 IdmaProperties::GetPropValBooleanById

Support for this method is optional.

4.2.33.3.1 Syntax

DmaRC **IdmaProperties::GetPropValBooleanById** (
 pDmaId *pPropertyId*,
 pDmaBoolean *pbBooleanValue*)

4.2.33.3.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be obtained. This identifier may be well-known or obtained from the meta-data.
<i>pbBooleanValue</i>	output	Returns the boolean value for the indicated property.

4.2.33.3.3 Description

Returns a boolean property value. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.33.3.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.4 IdmaProperties::GetPropValBooleanByIndex

Support for this method is optional.

4.2.33.4.1 Syntax

DmaRC **IdmaProperties::GetPropValBooleanByIndex** (
 DmaIndex32 *iIndex*,
 pDmaBoolean *pbBooleanValue*)

4.2.33.4.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	An index for the property that is to be obtained. This index may be obtained from the meta-data.
<i>pbBooleanValue</i>	output	Returns the boolean value for the indicated property.

4.2.33.4.3 Description

Returns a boolean property value. This is one of the basic data types used for property values. The property of interest is located using its index, which is defined by the object's Class Description.

4.2.33.4.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.5 IdmaProperties::GetPropValDateTimeById

Support for this method is optional.

4.2.33.5.1 Syntax

DmaRC **IdmaProperties::GetPropValDateTimeById** (
pDmaId *pPropertyId*,
ppDmaDateTime *ppDateTimeValue*)

4.2.33.5.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be obtained. This identifier may be well-known or obtained from the meta-data.
<i>ppDateTimeValue</i>	output	A pointer to the returned DateTime, which will be set to NULL if unsuccessful.

4.2.33.5.3 Description

Returns a DateTime property value, which is a specialization of a DMAString. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

This method, if successful, allocates a DmaDateTime and passes ownership to the caller.

4.2.33.5.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.6 IdmaProperties::GetPropValDateTimeByIndex

Support for this method is optional.

4.2.33.6.1 Syntax

DmaRC **IdmaProperties::GetPropValDateTimeByIndex** (
DmaIndex32 *iIndex*,
ppDmaDateTime *ppDateTimeValue*)

4.2.33.6.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be obtained. This index may be obtained from the meta-data.
<i>ppDateTimeValue</i>	output	A pointer to the returned DateTime, which will be set to NULL if unsuccessful.

4.2.33.6.3 Description

Returns a DateTime property value, which is a specialization of DMAString. The property of interest is located using its index, which is defined by the object's Class Description.

This method, if successful, allocates a `DmaDateTime` and passes ownership to the caller.

4.2.33.6.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.7 IdmaProperties::GetPropValFloat64ById

Support for this method is optional.

4.2.33.7.1 Syntax

`DmaRC IdmaProperties::GetPropValFloat64ById (`
`pDmaId pPropertyId,`
`pDmaFloat64 pfFloatValue)`

4.2.33.7.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to a string containing the identifier of the property to be obtained. This identifier may be well-known or obtained from the meta-data.
<i>pfFloatValue</i>	output	Returns the float value of the Property.

4.2.33.7.3 Description

Returns a 64 bit float property Value. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.33.7.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.8 IdmaProperties::GetPropValFloat64ByIndex

Support for this method is optional.

4.2.33.8.1 Syntax

DmaRC **IdmaProperties::GetPropValFloat64ByIndex** (
DmaIndex32 *iIndex*,
pDmaFloat64 *pfFloatValue*)

4.2.33.8.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be obtained. This index may be obtained from the meta-data.
<i>pfFloatValue</i>	output	Returns the float value of the Property.

4.2.33.8.3 Description

Returns a 64 bit float property Value. This is one of the basic data types used for property values. The property of interest is located using its index, which is defined by the object's Class Description.

4.2.33.8.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.

Name	Description
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.9 IdmaProperties::GetPropValldById

Support for this method is optional.

4.2.33.9.1 Syntax

DmaRC **IdmaProperties::GetPropValldById** (
 pDmald *pPropertyId*,
 pDmald *pIdValue*)

4.2.33.9.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be obtained. This identifier may be well-known or obtained from the meta-data.
<i>pIdValue</i>	output	Pointer to client-allocated storage, which the method will populate with the Id value

4.2.33.9.3 Description

Returns a Property Value in DMA Id form. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.33.9.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.10 IdmaProperties::GetPropValIdByIndex

Support for this method is optional.

4.2.33.10.1 Syntax

DmaRC **IdmaProperties::GetPropValIdByIndex** (
 DmaIndex32 *ilIndex*,
 pDmald *pldValue*)

4.2.33.10.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	An index for the property that is to be obtained. This index may be obtained from the meta-data.
<i>pldValue</i>	output	Pointer to client-allocated storage, which the method will populate with the Id value

4.2.33.10.3 Description

Returns a Property Value as a DMA Id, which is one of the basic data types used for property values. The property of interest is located using its index, which is defined by the object's Class Description.

4.2.33.10.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.11 IdmaProperties::GetPropValInteger32ById

Support for this method is optional.

4.2.33.11.1 Syntax

DmaRC **IdmaProperties::GetPropValInteger32ById** (
 pDmald *pPropertyId*,
 pDmaInteger32 *plIntegerValue*)

4.2.33.11.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to a string containing the identifier of the property to be obtained. This identifier may be well-known or obtained from the meta-data.
<i>plIntegerValue</i>	output	The integer value of the Property.

4.2.33.11.3 Description

Returns a 32 bit signed integer property value. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

4.2.33.11.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.12 IdmaProperties::GetPropValInteger32ByIndex

4.2.33.12.1 Support for this method is optional.

Syntax

DmaRC **IdmaProperties::GetPropValInteger32ByIndex** (
DmaIndex32 *iIndex*,
pDmaInteger32 *plIntegerValue*)

4.2.33.12.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be obtained. This index may be obtained from the meta-data.
<i>plIntegerValue</i>	output	The integer value of the Property.

4.2.33.12.3 Description

Returns a 32 bit signed integer property value. This is one of the basic data types used for property values. The property of interest is located using its index, which is defined by the object's Class Description.

4.2.33.12.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.13 IdmaProperties::GetPropValObjectById

Support for this method is optional.

4.2.33.13.1 Syntax

DmaRC **IdmaProperties::GetPropValObjectById** (
 pDmald *pPropertyId*,
 DMA_REFIID *riid*,
 pDmapv *ppObjectValue*)

4.2.33.13.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be obtained. This identifier may be well-known or obtained from the meta-data.
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppObjectValue</i>	output	An interface on the returned object, which will be set to NULL for unsuccessful return codes.

4.2.33.13.3 Description

Returns an object valued property. This operation will be successful only for properties that are of type "object". This allows properties to be containers, lists of property values, or complex types. The property of interest is located

using its identifier, which is either "well-known" or obtained from the meta-data.

If the returned object does not support the interface specified by riid, the operation will fail and the result code DMARC_BAD_INTERFACE will be returned. In case the operation is successful, the returned interface must be released using the method IUnknown::Release.

The returned object is obtained by reference. If there is an already-bound value for the object-valued property, an interface on that value is returned. If there is no already-bound value, and a value is defined, an appropriate DMA object is obtained, bound, and its selected interface returned. Sometimes, an existing object will be bound (i.e., for the dmaProp_This property). In other cases, a completely new object may be supplied, as for the default initial value of a never-set list-valued property. And at other times, a new scratchpad of an existing persistent object will be created and bound in conformance with the by-reference rules of the DMA object model.

An exception is made for enumeration objects -- all access to enumeration-valued properties return interfaces to a fresh enumeration object reset to the beginning of the sequence. Enumeration-valued properties always behave as if they are not already bound.

If this method is applied to a list or enumeration property, it may never return DMARC_VALUE_NOT_SET. That is, all other error conditions notwithstanding, an interface to a list or enumeration object must always be delivered, even though the list or enumeration may be empty.

4.2.33.13.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.14 IdmaProperties::GetPropValObjectByIndex

Support for this method is optional.

4.2.33.14.1 Syntax

```
DmaRC IdmaProperties::GetPropValObjectByIndex (
    DmaIndex32 ilIndex,
    DMA_REFIID riid,
    pDmapv ppIObjectValue)
```

4.2.33.14.2 Parameters

Name	Mode	Description
<i>ilIndex</i>	input	An index for the property that is to be obtained. This index may be obtained from the meta-data.
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppIObjectValue</i>	output	An interface on the returned object, which will be set to NULL for unsuccessful return codes.

4.2.33.14.3 Description

Returns an object valued property. This operation will be successful only for properties that are of type "object". This allows properties to be containers, lists of property values, or complex types. The property of interest is located using its index, which is defined by the object's Class Description.

If the returned object does not support the interface specified by *riid*, the operation will fail and the result code DMARC_BAD_INTERFACE will be returned. In case the operation is successful, the returned interface must be released using the method IUnknown::Release.

The returned object is obtained by reference. If there is an already-bound value for the object-valued property, an interface on that value is returned. If there is no already-bound value, and a value is defined, an appropriate DMA object is obtained, bound, and its selected interface returned. Sometimes, an existing object will be bound (i.e., for *dmaProp_This*). In other cases, a completely new object may be supplied, as for the default initial value of a never-set list-valued property. And at other times, a new scratchpad of an existing persistent object will be created and bound in conformance with the by-reference rules of the DMA object model.

An exception is made for enumeration objects -- all access to enumeration-valued properties return interfaces to a fresh enumeration object reset to the beginning of the sequence. Each time the property is accessed, enumeration-valued properties behave as if they are not already bound.

If this method is applied to a list or enumeration property, it may never return DMARC_VALUE_NOT_SET. That is, all other error conditions notwithstanding, an interface to a list or enumeration object must always be delivered, even though the list or enumeration may be empty.

4.2.33.14.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.15 IdmaProperties::GetPropValStringById

Support for this method is optional.

4.2.33.15.1 Syntax

DmaRC IdmaProperties::GetPropValStringById (
pDmald *pPropertyId*,
ppDmaString *ppStringValue*)

4.2.33.15.2 Parameters

Name	Mode	Description
<i>pPropertyId</i>	input	A pointer to the identifier of the property to be obtained. This identifier may be well-known or obtained from the meta-data.
<i>ppStringValue</i>	output	A pointer to the returned string, which will be set to NULL if unsuccessful.

4.2.33.15.3 Description

Returns a DMAString-valued property value. This is one of the basic data types used for property values. The property of interest is located using its identifier, which is either "well-known" or obtained from the meta-data.

This method, if successful, allocates a DmaString and passes ownership to the caller.

4.2.33.15.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.33.16 IdmaProperties::GetPropValStringByIndex

Support for this method is optional.

4.2.33.16.1 Syntax

DmaRC **IdmaProperties::GetPropValStringByIndex** (
DmaIndex32 *iIndex*,
ppDmaString *ppStringValue*)

4.2.33.16.2 Parameters

Name	Mode	Description
<i>iIndex</i>	input	An index for the property that is to be obtained. This index may be obtained from the meta-data.
<i>ppStringValue</i>	output	A pointer to the returned string, which will be set to NULL if unsuccessful.

4.2.33.16.3 Description

Returns a DMAString-valued property value, which is one of the basic data types used for property values. The property of interest is located using its index, which is defined by the object's Class Description.

This method, if successful, allocates a `DmaString` and passes ownership to the caller.

4.2.33.16.4 Return Values

Name	Description
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.2.34 IdmaRelationshipOrdering

This interface provides methods for manipulating the relative ordering of multiple relationship 'arrows' pointing out of a single Head or into a single Tail.

The methods below affect only the local scratch copies of the objects. To make these changes persistent, `IdmaConnection::ExecuteChange()` or `IdmaBatch::ExecuteChanges()` must be invoked on the relationship object.

4.2.34.1 IdmaRelationshipOrdering::SetOrdering

Support for this method is optional.

4.2.34.1.1 Syntax

```
DmaRC IdmaRelationshipOrdering::SetOrdering (
    Dmapv pIHeadRelationship,
    DmaInteger32 ulHeadWhere,
    Dmapv pITailRelationship,
    DmaInteger32 ulTailWhere)
```

4.2.34.1.2 Parameters

Name	Mode	Description
<i>pIHeadRelationship</i>	input	Another Relationship object of the same class as the current object. The current object and <i>pIHeadRelationship</i> object must have a common Head object.
<i>ulHeadWhere</i>	input	Position of the current object relative to <i>pIHeadRelationship</i> object.

Name	Mode	Description
<i>pITailRelationship</i>	input	Another Relationship object of the same class as the current object. The current object and pITailRelationship object must have a common Tail object.
<i>ulTailWhere</i>	input	Position of the current object relative to pITailRelationship object.

4.2.34.1.3 Description

This method manipulates the relative ordering of multiple relationship 'arrows' pointing out of a single Head or into a single Tail. It allows inserting before or after a specified existing relationship object, or first or last in the enumeration, or in an arbitrary document-space-defined order.

The values for ulHeadWhere and ulTailWhere are selected from the following set: DMA_POS_BEFORE inserts the current object immediately before the other object. DMA_POS_AFTER inserts the current object immediately after the other object. DMA_POS_FIRST inserts the current object as the first element of the enumeration. DMA_POS_LAST inserts the current object as the last element of the enumeration. DMA_POS_NO_CHANGE does not change the ordering (useful for setting an initial ordering). DMA_POS_DEFAULT leaves the choice of where the current object is inserted up to the doc space implementation. Only DMA_POS_DEFAULT is required to be supported if this interface is implemented. For DMA_POS_FIRST and DMA_POS_LAST, the 'other object' argument must be NULL.

In the context of containment, the Head and Tail properties of a Referential Containment Relationship object, for instance, are objects of class Containable and Container, respectively. The reflective property for Head is Containers in an object of class Containable. The reflective property for Tail is Containees in an object of class Container. Thus, invoking this method on a Referential Containment Relationship object after setting the Head and Tail properties will insert the relationship object into the Containers and Containees enumerations of the related Containable and Container objects.

Similarly, the Head and Tail properties of a Direct Containment Relationship object are objects of class Containable and Container, respectively. The reflective property pairs are Head and Parent, and Tail and Children. Note that Parent is a singleton Direct Containment Relationship object, rather than an enumeration, due to the 1:N nature of direct containment. In this case, the only valid values for pIHeadRelationship and ulHeadWhere are null and DMA_POS_DEFAULT, respectively.

If multiple calls are made to this method without intervening calls on ExecuteChange or ExecuteChanges, each such call completely replaces the intended effect of the preceding call. Thus, when ExecuteChange or

ExecuteChanges is invoked, the only ordering change to the persistent store will be those reflecting the most recent call to SetOrdering.

4.2.34.1.4 Return Values

Name	Description
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_CONFLICTING_OPERATION	The call on this primary intentional method logically conflicts with a prior call on a different primary intentional method.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_READ_ONLY	Method failed because an object or property is read-only.
DMARC_UNSUPPORTED_ORDERING	An unacceptable value was provided for an ordering parameter.

4.2.35 IdmaResultSet

The IdmaResultSet interface provides the methods specific to a result set object, such as asking when search results are available, retrieving those results, terminating and restarting a search.

4.2.35.1 IdmaResultSet::GetNextResultRow

This method must be supported in every implementation of this interface.

4.2.35.1.1 Syntax

DmaRC **IdmaResultSet::GetNextResultRow** (
DMA_REFIID *riid*,
pDmapv *ppIResultRow*)

4.2.35.1.2 Parameters

Name	Mode	Description
<i>riid</i>	input	The Id of the desired interface on the result row object.
<i>ppIResultRow</i>	output	Location in which to return an interface to the result row object

4.2.35.1.3 Description

This method returns the next query result row (dmaClass_QueryResultRow), if any.

4.2.35.1.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_MAX_ROWS	A result set was truncated because it exceeded the maximum result items parameter.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_OK	(S_OK) Success.
DMARC_QUERY_AT_END	The query ran to successful completion without encountering any constraints on time limits, number of rows generated, or any other constraints, and there are no more result rows that satisfy the query. A result row is not returned. No more result rows will be returned unless ReExecuteQuery is executed.
DMARC_RESULTS_TRUNCATED	The result set was truncated explicitly by a call on IdmaResultSet::TerminateResults. No further rows will be returned unless ReExecuteQuery is executed. This is a failure code - no row is returned.
DMARC_TIMEOUT	A time limit has been exceeded.

4.2.35.2 IdmaResultSet::IsResultReady

Support for this method is optional.

4.2.35.2.1 Syntax

DmaRC **IdmaResultSet::IsResultReady** (
pDmaBoolean *pblsReady*)

4.2.35.2.2 Parameters

Name	Mode	Description
<i>pblsReady</i>	output	Returns DMA_TRUE if the search results are ready to be delivered.

4.2.35.2.3 Description

Poll to see if the results are ready to be delivered. When the IdmaScope::ExecuteSearch method returns, the search operation may not be complete (e.g., the result set may not be ready to deliver its results). The *pblsReady* parameter will be set to the value DMA_FALSE if the results are not ready, and will be set to DMA_TRUE if at least one result set item is ready to be delivered.

In more detail, DMA_TRUE is returned if at least one result set item can be returned by GetNextResultRow without blocking, or if all the rows satisfying the search criteria have been returned, or because the result set has been truncated (see the MaximumResultItems property of the query object) and the last available row has already been returned by GetNextResultRow.

This method will not return FALSE indefinitely if there are any results rows that can be generated.

Note that support of this method is optional. A document space may choose to implement synchronous searches (ExecuteSearch does not return until all results are ready). If this method is supported, a document space may choose to generate result items as a side effect of calls upon it.

4.2.35.2.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_RESULTS_TRUNCATED	The result set was truncated explicitly by a call on IdmaResultSet::TerminateResults. No further rows will be returned unless ReExecuteQuery is executed. This is a failure code - no row is returned.

4.2.35.3 IdmaResultSet::ReExecuteQuery

This method must be supported in every implementation of this interface.

4.2.35.3.1 Syntax

DmaRC **IdmaResultSet::ReExecuteQuery** (
Dmapv *pICallback*)

4.2.35.3.2 Parameters

Name	Mode	Description
<i>pICallback</i>	input	Pointer to a callback interface whose ReportProgress method will optionally be called at regular intervals during this function call. This pointer may be NULL.

4.2.35.3.3 Description

This method performs processing necessary for reexecution of the query against the persistent store, so that GetNextResultRow can be called to generate a fresh sequence of query result rows.

4.2.35.3.4 Return Values

Name	Description
DMARC_ABORT	The progress callback indicated that the method should be aborted.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_OK	(S_OK) Success.

4.2.35.4 IdmaResultSet::TerminateResults

This method must be supported in every implementation of this interface.

4.2.35.4.1 Syntax

DmaRC **IdmaResultSet::TerminateResults** ()

4.2.35.4.2 Description

Terminate delivery of result rows for this result set.

After a successful call to TerminateResults, any of the non-inherited IdmaResultSet methods return DMARC_RESULTS_TRUNCATED until ReExecuteQuery is called.

4.2.35.4.3 Return Values

Name	Description
DMARC_OK	(S_OK) Success.
DMARC_RESULTS_TRUNCATED	The result set was truncated explicitly by a call on IdmaResultSet:: TerminateResults. No further rows will be returned unless ReExecuteQuery is executed. This is a failure code - no row is returned.

4.2.36 IdmaScope

The IdmaScope interface provides methods specific to a DMA scope, such as executing a query.

4.2.36.1 IdmaScope::ExecuteSearch

This method must be supported in every implementation of this interface.

4.2.36.1.1 Syntax

DmaRC **IdmaScope::ExecuteSearch** (
Dmapv *pIQueryObject*,
Dmapv *pICallback*,

DmaBoolean *bRequestElimination*,
DMA_REFIID *riidResultSet*,
pDmapv *pplResultSet*)

4.2.36.1.2 Parameters

Name	Mode	Description
<i>plQueryObject</i>	input	This is the query object that specifies the query to be performed.
<i>plCallback</i>	input	This is a pointer to a callback interface whose ReportProgress method will optionally be called at regular intervals during this function call. This pointer may be NULL.
<i>bRequestElimination</i>	input	If TRUE, this requests the scope to eliminate classes, properties, and operators it doesn't fully understand using the rules of three valued logic. If the scope supports this capability, it will comply. If FALSE, an error will be returned if the scope encounters a class, property, or operator it doesn't understand.
<i>riidResultSet</i>	input	The desired interface on the result set object.
<i>pplResultSet</i>	output	The result set object returned by this method. The query result items are obtained by invoking methods on this object.

4.2.36.1.3 Description

The ExecuteSearch method processes the Query Object and internalizes the information contained therein. Neither the Query Object nor any of its subobjects are altered in any way by ExecuteSearch.

If the underlying scope is a merged scope, then, assuming preliminary processing succeeds, this function passes either the Query Object, or a newly constructed Query Object to the IdmaScope:: ExecuteSearch method of each component scope. If any of these calls returns an error, an error is returned.

In the case that a new Query Object is passed to a component scope, the new Query Object is not necessarily isomorphic to the Query Object passed in to this method. Examples of when this could occur include (1) three valued elimination is performed on the information in the original Query Object to develop the new copy, and/or (2) the Order By clause is added or extended in the new copy.

No internal pointers are kept to the Query Object or any of its descendant nodes. Once ExecuteSearch returns, changes to the Query Object or its descendant nodes will have no effect on the subsequent development of result rows during query execution.

If `ExecuteSearch` is successful, a Result Set object is returned. The `IdmaResultSet:: GetNextResultRow` method on the Result Set object is used to obtain each result row. The `IdmaResultSet:: ReExecuteQuery` method on the Result Set object may be used to terminate the current query, and prepare for reexecution of the query (without calling `ExecuteSearch` again), etc. The `IdmaResultSet:: TerminateResults` method on the Result Set object may be used to cancel further execution of the query. The `TerminateResults` method can be called asynchronously by a second thread in a multi-threaded application while the first thread is blocked internal to `GetNextResultRow` awaiting a result row to be returned. Multiple Result Set objects can be generated from a scope object. The Result Set objects are independent of each other, and, in a multi-threaded application, can be used concurrently.

It is recommended but not required that implementations defer most or all of the document space scope processing above and beyond the processing of the Query Object to the Result Set object, in order to make `ExecuteSearch` complete faster. If this is done, a multi-threaded application would then be more responsive to asynchronously canceling the potentially long running part of the query (i.e., generation of result rows) by calling `IdmaResultSet:: TerminateResults` on the Result Set object.

Clients desiring proper errors for their malformed Query Objects should pass `DMA_FALSE` for the `bRequestElimination` parameter to `ExecuteSearch`.

If `DMA_FALSE` is passed for `bRequestElimination`, and the query contains any undefined elements, `ExecuteSearch` will return an error.

If `DMA_TRUE` is passed for `bRequestElimination`, and the scope does not support three valued elimination, `DMARC_3VE_NOT_SUPPORTED` is returned.

If `bRequestElimination` is `DMA_TRUE`, and the query contains undefined elements, the scope performs three valued elimination as needed on the query object relative to its own metadata to develop a new copy of the information in the query object, which does not contain the undefined elements. Processing of the derived query object then proceeds.

If the scope is a merged scope, it delivers the query or a massaged copy of it to each component scope. For each component scope, the merged scope determines whether there are any undefined classes, properties, etc., in the possibly revised Query Object relative to the metadata of the component scope. If there are any undefined elements, and the component scope does not support three valued elimination, then the merged scope must perform three valued elimination to develop a revised query object which contains only elements defined relative to the metadata of the component scope. This insures that a query which is valid relative to the metadata of the merged scope will never fail because it contains undefined elements relative to any component scope. This behavior is independent of the setting of the `Has Elimination` property on the merged scope.

4.2.36.1.4 Return Values

Name	Description
DMARC_3VE_NOT_SUPPORTED	The current scope does not support three valued elimination.
DMARC_ABORT	The progress callback indicated that the method should be aborted.
DMARC_BAD_CLASSID	The supplied identifier does not reference an available class of object.
DMARC_BAD_COLLATION_SEQUENCE	The specified collation sequence is not known or supported by the scope.
DMARC_BAD_FROM_EXPRESSION	The query contains an invalid From Expression.
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_OPERAND	An operand is incorrect in datatype, form, or class.
DMARC_BAD_OPERATOR	An operator ID is not defined in the metadata of the current scope.
DMARC_BAD_ORDERBY_ELEMENT	Illegal 'order by' list element.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_BAD_SC_OCCUR	The value of a query node's searchable class occurrence property is not present in the query.
DMARC_BAD_SELECT_ELEMENT	Illegal select list element.
DMARC_BAD_SUBQUERY_DATATYPE	The datatype of the query root node of an "IN" subquery does not match that of the property in the select list.
DMARC_BAD_SUBQUERY_SELECT	The select list of a subquery contains an inappropriate number of elements
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_CLASS_NOT_SEARCHABLE	A class referred to in the From Expression is not searchable.
DMARC_CLASSES_NOT_JOINABLE	A particular join operator in the query is supported, but the operator does not support the join between the particular operands specified in the query.
DMARC_DISTINCT_NOT_SUPPORTED	The scope does not support the request for distinct result rows.
DMARC_EXTRA_OPERANDS	A query node has extra operands.
DMARC_MISSING_OPERANDS	A query node is missing one or more operands.
DMARC_OK	(S_OK) Success.
DMARC_PROPERTY_NOT_ORDERABLE	A property referred to in the query order by list is not orderable.
DMARC_PROPERTY_NOT_SEARCHABLE	A property referred to in the Query Expression is not searchable.

Name	Description
DMARC_PROPERTY_NOT_SELECTABLE	A property referred to in the Selections list is not selectable.
DMARC_QUERY_TREE_LOOP	There is a circular reference loop or a shared node in the query.
DMARC_REQUIRED_VALUE_ABSENT	A required property value has not been set.
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.

4.2.36.2 IdmaScope::GetOperatorDescription

This method must be supported in every implementation of this interface.

4.2.36.2.1 Syntax

DmaRC **IdmaScope::GetOperatorDescription** (
 pDmald *pOperatorId*,
 DMA_REFIID *riid*,
 pDmapv *ppOperatorDesc*)

4.2.36.2.2 Parameters

Name	Mode	Description
<i>pOperatorId</i>	input	The Id of the operator whose description is desired.
<i>riid</i>	input	The desired interface on the operator description object.
<i>ppOperatorDesc</i>	output	The operator description object returned.

4.2.36.2.3 Description

This method can be used to get the operator description object (an instance of dmaClass_QueryOperatorDescription) for a particular object. This can be useful in building or analyzing query expression hierarchies.

4.2.36.2.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.37 IdmaScopeFactory

This is the specialized interface for scope factories, optionally supported on the DMA System object. It provides methods for creating lists usable with the

Scope Factory and it provides the CreateScope method for merging multiple component scopes into a single logical scope for unified searching.

4.2.37.1 IdmaScopeFactory::CreateScope

This method must be supported in every implementation of this interface.

4.2.37.1.1 Syntax

```
DmaRC IdmaScopeFactory::CreateScope (
    Dmapv pIScopeList,
    DmaInteger32 ICombinationRule,
    DMA_REFIID riid,
    pDmapv ppIScope)
```

4.2.37.1.2 Parameters

Name	Mode	Description
<i>pIScopeList</i>	input	A DMA List containing the list of scopes that will make up the merged scope. Each object in the list must provide the interfaces required for a DMA Scope object.
<i>ICombinationRule</i>	input	Combination Rule used to merge the scopes in the input scope list.
<i>riid</i>	input	Interface Identifier for the desired interface on resulting Scope object.
<i>ppIScope</i>	output	Returns the selected interface for the merged scope object.

4.2.37.1.3 Description

Creates a merged scope object that is a combination of the supplied list of scopes. Each object on the list must provide all of the interfaces required for a dmaClass_Scope type object. The well-known properties specified for dmaClass_Scope must be supported. If any objects on the list are found to be invalid, the method will fail and the DMARC_BAD_SCOPE result code will be returned. If the merge operation is unsuccessful an appropriate result code will be returned and *ppIScope will be set to NULL. If a non-NULL value is returned in *ppIScope, it will be an interface to a valid scope object

There are two defined rules for combining the input scope objects. They are DMA_COMBINATION_INTERSECTION and DMA_COMBINATION_UNION.

If the returned object does not support the interface specified by riid, the operation will fail and the result code DMARC_BAD_INTERFACE will be returned. If the operation is successful, the returned interface pointer is already reserved on behalf of the caller in accordance with the rules for delivery of interfaces as outputs of COM methods. (See IUnknown::QueryInterface.).

CreateScope must be able to handle more than one component scope in the DMA List of Objects identified by pIScopeList.

4.2.37.1.4 Return Values

Name	Description
DMARC_ALIAS_CONFLICT	A class, operator or property in one component scope unifies with more than one such entity in another component scope.
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NO_COLLATIONS	Merged scope creation failed because there would be no supported collation sequences.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_OK	(S_OK) Success.
DMARC_OPERAND_MERGE_CONFLICT	The merge would create an invalid operand description.
DMARC_OPERATOR_MERGE_CONFLICT	An operator could not be merged because it has conflicting signatures in component scopes.
DMARC_PROPERTY_MERGE_CONFLICT	A property could not be merged because there is a conflict of conflicting datatype or cardinality in component scopes.

4.2.37.2 IdmaScopeFactory::CreateScopeList

This method must be supported in every implementation of this interface.

4.2.37.2.1 Syntax

```
DmaRC IdmaScopeFactory::CreateScopeList (
    DMA_REFIID riid,
    pDmapv ppIObject)
```

4.2.37.2.2 Parameters

Name	Mode	Description
<i>riid</i>	input	Interface Identifier for the desired interface on the resulting List of Scopes object.
<i>ppIObject</i>	output	Returns a pointer for the requested interface for the initial List of Scopes object, *ppIObject will be NULL if CreateScopeList is unsuccessful.

4.2.37.2.3 Description

This will create a new, empty but editable DMA List object of class `dmaClass_ListOfObject`. The list will be empty and the list object's properties will be set with default values. The list will be populated by using the `IdmaEditList` and `IdmaEditListOfObject` interface. The list supports at least those interfaces specified as required for a class `dmaClass_ListOfObject` object.

If the requested object does not support the interface specified by `riid`, the operation will fail and the result code `DMARC_BAD_INTERFACE` will be returned. If the operation is successful, the returned interface pointer is already reserved on behalf of the caller, in accordance with the rules for delivery of interfaces as outputs of COM methods. (See `IUnknown::QueryInterface`).

4.2.37.2.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_OK	(S_OK) Success.

4.2.38 IdmaServiceRegistry

Provides access to registry elements that maintain information about the implementation of DMA Service Objects

4.2.38.1 IdmaServiceRegistry::FindNextRegistryElement

This method must be supported in every implementation of this interface.

4.2.38.1.1 Syntax

```
DmaRC IdmaServiceRegistry::FindNextRegistryElement (
    pDmald pPrevRegistrationId,
    pDmald pServiceObjectIdFilter,
    pDmald pServiceObjectTypeIdFilter,
    pDmaInteger32 pCharSetEncodingIdFilter,
    pDmaBoolean pbPublishedFilter,
    pDmald pRegistrationId,
    pDmald pServiceObjectId,
    pDmald pServiceObjectTypeId,
    ppDmaString ppProfile,
    pDmald pModuleTypeId,
```


ppDmaString ppModuleLocation,
pDmaInteger32 pCharSetEncodingId,
pDmaBoolean pbPublished)

4.2.38.1.2 Parameters

Name	Mode	Description
<i>pPrevRegistrationId</i>	input	A reference to the registration Id of an existing Registry element or NULL.
<i>pServiceObjectIdFilter</i>	input	A reference to the Id of a Service Object or NULL. If non-NULL, this method will only return registration elements with Service Object Ids that match the dereferenced value of this parameter.
<i>pServiceObjectTypeIdFilter</i>	input	A reference to the Id of a Service Object Type or NULL. If non-NULL, this method will only return registration elements with Service Object Type Ids that match the dereferenced value of this parameter.
<i>pCharSetEncodingIdFilter</i>	input	A reference to a Character Set Encoding Id or NULL. If non-NULL, this method will only return registration elements with Character Set Encoding Ids that match the dereferenced value of this parameter.
<i>pbPublishedFilter</i>	input	A reference to a boolean or NULL. If non-NULL, this method will only return registration elements with Published values that match the dereferenced value of this parameter.
<i>pRegistrationId</i>	output	(optional) The registration Id of a previously registered registry element. This parameter uniquely identifies the registry element accessed by this method.
<i>pServiceObjectId</i>	output	(optional) The Id of a Service Object that is registered with the System.
<i>pServiceObjectTypeId</i>	output	<i>pServiceObjectTypeId</i> output (optional) Returns the Type Id for the service object.
<i>ppProfile</i>	output	(optional) Returns a copy of the profile string that is supplied to the Service Object when it is activated.
<i>pModuleTypeId</i>	output	(optional) A DmaId value that specifies of the type of Module used to implement the Service Object, e.g. a WIN32 DLL.
<i>ppModuleLocation</i>	output	(optional) Returns a string that identifies the location of the Module.

Name	Mode	Description
<i>pCharSetEncodingId</i>	output	(optional) Returns the character set encoding that is supported by this service object.
<i>pbPublished</i>	output	(optional) Returns a boolean indicating whether this registration element has been published.

4.2.38.1.3 Description

Returns the selected registration information for the registry element "following" the PrevRegistrationId in whatever order the Service Registry implementation maintains registry element entries.

The list of registered service element entries can be browsed by calling this method initially with a NULL value for pPrevRegistrationId. Subsequent methods calls would specify pRegistrationId as pPrevRegistrationId while this method returns DMARC_OK. A DMARC_BAD_PARAMETER value will be returned if pPrevRegistrationId is not NULL and a registry entry with that RegistrationId does not exist. A DMARC_NOT_FOUND value will be returned if all parameters are valid and no registry entry is found that meets the specified criteria.

This method supports filtering the list of registered service element entries based upon a combination of Service Object Id, Service Object Type Id, Character Set Encoding Id and bPublished through the specification of non-NULL values for any of the corresponding filter input parameters. Providing specific values for Service Object Id, Service Object Type Id, Character Set Encoding Id and a NULL value for pPrevRegistrationId is guaranteed to return at most one registry element.

If an output parameter reference is NULL, the value for that information will not be returned.

4.2.38.1.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_FOUND	Requested item not found.
DMARC_OK	(S_OK) Success.

4.2.38.2 IdmaServiceRegistry::GetRegistryElement

This method must be supported in every implementation of this interface.

4.2.38.2.1 Syntax

DmaRC IdmaServiceRegistry::GetRegistryElement (
 pDmaId *pRegistrationId*,
 pDmaId *pServiceObjectId*,
 pDmaId *pServiceObjectTypeId*,
 ppDmaString *ppProfile*,
 pDmaId *pModuleTypeId*,
 ppDmaString *ppModuleLocation*,
 pDmaInteger32 *plCharSetEncodingId*,
 pDmaBoolean *pbPublished*)

4.2.38.2.2 Parameters

Name	Mode	Description
<i>pRegistrationId</i>	input	The registration Id of a previously registered registry element. This parameter uniquely identifies the registry element accessed by this method.
<i>pServiceObjectId</i>	output	(optional) The Id of a Service Object that is registered with the System.
<i>pServiceObjectTypeId</i>	output	(optional) Returns the type the service object.
<i>ppProfile</i>	output	(optional) Returns a copy of the profile string that is supplied to the Service Object when it is activated.
<i>pModuleTypeId</i>	output	(optional) A DmaId GUID value that specifies of the type of Module is used to implement the Service Object, e.g. WIN32 DLL.
<i>ppModuleLocation</i>	output	(optional) Returns a string that identifies the location of the Module.
<i>plCharSetEncodingId</i>	output	(optional) Returns the character set encoding that is supported by this service object.
<i>pbPublished</i>	output	(optional) Returns a boolean indicating whether this registration element has been published.

4.2.38.2.3 Description

Returns the selected registration information for the identified registry element. If an output parameter reference is NULL, the value for that information will not be returned.

4.2.38.2.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.

Name	Description
DMARC_NOT_FOUND	Requested item not found.
DMARC_OK	(S_OK) Success.

4.2.38.3 IdmaServiceRegistry::PublishRegistryElement

Support for this method is optional.

4.2.38.3.1 Syntax

DmaRC **IdmaServiceRegistry::PublishRegistryElement**

(pDmaId *pRegistrationId*)

4.2.38.3.2 Parameters

Name	Mode	Description
<i>pRegistrationId</i>	input	The registration Id of a previously registered registry element. This parameter uniquely identifies the registry element that will be published by this method.

4.2.38.3.3 Description

This method publishes a Service Registry element. Upon successful completion of this method, the registry element's bPublished attribute will have a value of DMA_TRUE.

A Service Object that is initially registered is known only at the point of presence where it is registered. Publishing the registration entry makes the service element known to all points-of-presence that can reach the point-of-service having the service object.

4.2.38.3.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_FOUND	Requested item not found.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.

4.2.38.4 IdmaServiceRegistry::RegisterServiceElement

Support for this method is optional.

4.2.38.4.1 Syntax

DmaRC IdmaServiceRegistry::RegisterServiceElement (
pDmaId pServiceObjectId,
pDmaId pServiceObjectTypeId,
pDmaString pProfile,
pDmaId pModuleTypeId,
pDmaString pModuleLocation,
DmaInteger32 lCharSetEncodingId,
pDmaId pRegistrationId)

4.2.38.4.2 Parameters

Name	Mode	Description
<i>pServiceObjectId</i>	input	The DmaId to use as the Service Object's instance Id. If NULL, a DmaId GUID value will be assigned by the RegisterServiceElement operation
<i>pServiceObjectTypeId</i>	input	The Service Object's type, e.g. a System, DocSpace, ScopeFactory, TextOrdering, or other registry-supported type.
<i>pProfile</i>	input	(optional) The profile string that will be supplied to the Service Object when it is activated via its GetServiceObject function.
<i>pModuleTypeId</i>	input	Specifies the type of Module used to implement the Service Object, e.g. a Win32 DLL.
<i>pModuleLocation</i>	input	The location of the Module containing the service-object implementation program code.
<i>lCharSetEncodingId</i>	input	The character set encoding that is supported by this implementation of the service object.
<i>pRegistrationId</i>	output	Returns the registration that uniquely identifies the registry element registered by this method.

4.2.38.4.3 Description

This method adds an unpublished registry element to the Service Registry database.

The combination of *ServiceObjectId*, *ServiceObjectTypeId* and *CharSetEncodingId* must be unique across all registered elements. An attempt to register an element that would violate this uniqueness invariant condition will result in a DMARC_NOT_UNIQUE return code.

DMA 1.0 defines well-known *ServiceType* values *dmaServiceType_System*, *dmaServiceType_DocSpace*, *dmaServiceType_ScopeFactory*, *dmaServiceType_OIIDParser*, *dmaServiceType_TextOrdering* for the Service Objects of the named types, respectively.

The ServiceObjectId is an instance Id associated with the service object and is defined as the SystemId for System service objects, the DocSpaceId for DocSpace service objects, or the TextOrderingId (Collation Id) for TextOrdering service objects. For an OIIDParser, the ServiceObjectId is the unique dmaService_ OIIDParser value. Any module registered as a dmaServiceType_OIIDParser service must support the IdmaOIID interface for OIID strings in the character set encoding for which it is registered.

4.2.38.4.4 Return Values

Name	Description
DMARC_ACCESS_DENIED	(E_ACCESSDENIED) The requester has insufficient access rights to perform the requested operation.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.
DMARC_OK	(S_OK) Success.

4.2.38.5 IdmaServiceRegistry::RemoveRegistryElement

Support for this method is optional.

4.2.38.5.1 Syntax

DmaRC **IdmaServiceRegistry::RemoveRegistryElement** (
pDmaId *pRegistrationId*)

4.2.38.5.2 Parameters

Name	Mode	Description
<i>pRegistrationId</i>	input	The registration Id of a previously registered registry element. This parameter uniquely identifies the registry element which will be removed by this method.

4.2.38.5.3 Description

This method removes a registry element from the Service Registry database. If the registry element was published, it is unpublished prior to removal. The Registration Id is guaranteed not to be reused by the Service Registry.

4.2.38.5.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_FOUND	Requested item not found.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.

4.2.38.6 IdmaServiceRegistry::UnpublishRegistryElement

Support for this method is optional.

4.2.38.6.1 Syntax

DmaRC **IdmaServiceRegistry::UnpublishRegistryElement** (
pDmald *pRegistrationID*)

4.2.38.6.2 Parameters

Name	Mode	Description
<i>pRegistrationID</i>	input	The registration Id of a previously registered registry element. This parameter uniquely identifies the registry element which will be removed by this method.

4.2.38.6.3 Description

This method unpublishes a Service Registry element. Upon successful completion of this method, the registry element's bPublished attribute will have a value of DMA_FALSE.

4.2.38.6.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_FOUND	Requested item not found.
DMARC_NOT_PUBLISHED	The service registry entry is not currently published.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.

4.2.39 IdmaStream

Provides read-only access to DMA content via an octet-stream.

4.2.39.7 IdmaStream::ReadStreamData

This method must be supported in every implementation of this interface.

4.2.39.7.1 Syntax

```
DmaRC IdmaStream::ReadStreamData (
    pDmaBinarypBuffer,
    DmaUInteger32 ulBytesToRead,
    pDmaUInteger32 pulBytesRead)
```

4.2.39.7.2 Parameters

Name	Mode	Description
<i>pBuffer</i>	output	A pointer to the buffer to be read into
<i>ulBytesToRead</i>	input	Number of bytes to read from the stream
<i>pulBytesRead</i>	output	Pointer to number of bytes actually read.

4.2.39.7.3 Description

Reads the designated number of bytes into the buffer. The supplied buffer must be at least *ulBytesToRead* bytes in size.

Number of bytes read is returned in *pulBytesRead*, even if an error is returned. The seek pointer is adjusted by the number of bytes actually read.

If the end of stream is reached during the read, the number of bytes actually read may be less than requested and the result code `DMARC_SHORT_READ` is returned.

4.2.39.7.4 Return Values

Name	Description
<code>DMARC_BAD_PARAMETER</code>	(E_INVALIDARG) Invalid input parameter.
<code>DMARC_DEVICE_ERROR</code>	An error has occurred reading or writing a hardware device.
<code>DMARC_OK</code>	(S_OK) Success.
<code>DMARC_SHORT_READ</code>	(Warning) End of stream was encountered before the requested number of bytes was read.

4.2.39.8 IdmaStream::SetStreamPosition

Support for this method is optional.

4.2.39.8.1 Syntax

DmaRC **IdmaStream::SetStreamPosition** (
DmaInt64 *liDisplacement*,
DmaInteger32 *IRelativeTo*,
pDmaUInt64 *puliNewPosition*)

4.2.39.8.2 Parameters

Name	Mode	Description
<i>liDisplacement</i>	input	Number of bytes of displacement
<i>IRelativeTo</i>	input	DMA_REL_CURRENT if displacement is to be treated relative to the current position of the stream, DMA_REL_BEGINNING if it is relative to the beginning of the stream, or DMA_REL_END if it is relative to the end of the stream.
<i>puliNewPosition</i>	output	Pointer to value of the seek pointer, May be omitted, indicating client is not interested in this value

4.2.39.8.3 Description

Adjusts the location of the seek pointer on the stream. The seek pointer can be moved relative to its current position, set to an absolute location (relative to beginning of the stream)), or set relative to the end of the stream

If *liDisplacement* is greater than zero, the seek pointer is positioned toward the end of the stream. If *liDisplacement* is less than zero, the seek pointer is positioned toward the beginning of the stream. The seek pointer cannot be moved beyond the end of the stream or before the beginning of the stream. Attempting to do so will result in the seek pointer being set to either the beginning of the stream or the end of the stream.

If the stream cannot be positioned as requested, the error DMARC_POSITION_NOT_ALLOWED should be returned. For example, the stream may be "forward only" and the combination of *liDisplacement* and *IRelativeTo* may reference a point in the stream which has already been passed.

4.2.39.8.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_DEVICE_ERROR	An error has occurred reading or writing a hardware device.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.

Name	Description
DMARC_OK	(S_OK) Success.
DMARC_POSITION_NOT_ALLOWED	The stream cannot be positioned as requested.

4.2.40 IdmaSystem

This is the specialized interface for the DMA System object. It provides access to Document Spaces and methods that deal with multiple Document Spaces. It also provides some infrastructure services, such as access to the memory management mechanism.

4.2.40.1 IdmaSystem::ConnectDocSpace

This method must be supported in every implementation of this interface.

4.2.40.1.1 Syntax

```
DmaRC IdmaSystem::ConnectDocSpace (
    pDmald pDocSpaceId,
    pDmaString pLocaleName,
    DMA_REFIID riid,
    pDmapv ppIDocSpace)
```

4.2.40.1.2 Parameters

Name	Mode	Description
<i>pDocSpaceId</i>	input	The Dmald which identifies the document space for which an object is to be instantiated.
<i>pLocaleName</i>	input	The name of the locale to be used by the returned document space instance. If this parameter is NULL, the document space is to employ its default locale. If this parameter is non-NULL and does not correspond to a locale supported by the document space, the returned instance will utilize the document space's default locale.
<i>riid</i>	input	Interface Identifier for the desired interface on the resulting DMA object
<i>ppIDocSpace</i>	output	Reference to the requested interface for the DocSpace object; NULL if the operation is unsuccessful.

4.2.40.1.3 Description

Returns an interface to a (usually unauthenticated) DMA DocSpace object representing the identified DMA Document Space. If the operation succeeds, a new DocSpace object is instantiated, representing a point-of-access view of

the identified document space. Each DocSpace interface returned by `IdmaSystem::ConnectDocSpace` is for a distinct instance of a DMA DocSpace object (in the same sense that `EnumerationOfObject` delivers distinct instances according to the DMA Object Model).

In general, the `IdmaAuthentication` interface must be exercised on the returned DMA DocSpace object before the object can be used for anything other than inspecting non-object valued properties.

4.2.40.1.4 Return Values

Name	Description
DMARC_BAD_DOC_SPACE	There is no available document space with the specified <code>Dmald</code>
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_OK	(S_OK) Success.
DMARC_UNEXPECTED	(E_UNEXPECTED) An unexpected error occurred.

4.2.40.2 IdmaSystem::EnumerateDocSpaces

This method must be supported in every implementation of this interface.

4.2.40.2.1 Syntax

```
DmaRC IdmaSystem::EnumerateDocSpaces (
    DMA_REFIID riid,
    pDmapv ppIEnumOfDocSpace)
```

4.2.40.2.2 Parameters

Name	Mode	Description
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppIEnumOfDocSpace</i>	output	Returns a reference to the requested interface or NULL if unsuccessful.

4.2.40.2.3 Description

The `IdmaSystem::EnumerateDocSpaces` function returns an interface (typically `IdmaEnumOfObject`) to a DMA object which the caller can use to enumerate DMA Document Space objects accessible via the System object. The DocSpace objects delivered by the enumerator are in the same state as if `ConnectDocSpace` had been called, using the locale and character set encod-

ing in effect for the System object. Therefore, in general, the IdmaAuthentication interface must be exercised on the returned objects before they can be used for anything other than inspecting the DocSpace's non-object valued properties.

If the returned object does not support the interface specified by riid, the operation will fail and the result code DMARC_BAD_INTERFACE will be returned. In case the operation is successful, the returned interface must be released using the method IUnknown::Release.

4.2.40.2.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_OK	(S_OK) Success.

4.2.40.3 IdmaSystem::GetResultCodeDescription

This method must be supported in every implementation of this interface.

4.2.40.3.1 Syntax

DmaRC **IdmaSystem::GetResultCodeDescription** (
DmaRC *rcResultCode*,
ppDmaString *ppResultCodeDesc*)

4.2.40.3.2 Parameters

Name	Mode	Description
<i>rcResultCode</i>	input	A result code that may be returned from a DMA method.
<i>ppResultCodeDesc</i>	output	Returns a string that describes the selected result code.

4.2.40.3.3 Description

Returns a string that contains a textual description of the result code that is passed in to the function. The description will be given in the operative locale for the System object.

4.2.40.3.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_RC	Invalid Result Code.
DMARC_OK	(S_OK) Success.

4.2.41 IdmaSystemManager

The System Manager interface is an application's access to the DMA information that is visible to and available at a point of access. It is also used by DMA-compliant applications to enumerate and connect to registered systems.

4.2.41.1 IdmaSystemManager::ConnectSystem

This method must be supported in every implementation of this interface.

4.2.41.1.1 Syntax

DmaRC **IdmaSystemManager::ConnectSystem** (
 pDmaId *pSystemId*,
 pDmaString *pLocaleName*,
 DMA_REFIID *riid*,
 pDmapv *ppISystem*)

4.2.41.1.2 Parameters

Name	Mode	Description
<i>pSystemId</i>	input	The System Id which uniquely identifies the system configuration to connect to.
<i>pLocaleName</i>	input	The name of the locale to be used by the returned system object. If this parameter is NULL, the system object is to employ its default locale. If this parameter is non-NULL and does not correspond to a locale supported by this system object, the system object will utilize its default locale.
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppISystem</i>	output	Returns a reference to the requested interface for the System object, which will be NULL if unsuccessful.

4.2.41.1.3 Description

Returns an interface that provides a (usually unauthenticated) DMA System object representing the indicated DMA System. If the operation succeeds, a

new System Object is returned, representing a point-of-access view of the identified system. Each System Object returned by `IdmaSystemManager::ConnectSystem` is a distinct instance.

In general, the `IdmaAuthentication` interface must be exercised on the returned DMA System object before the object can be used for anything other than inspecting the system's non-object valued properties.

There is a small well defined set of DMA return codes returned from this method. There is no DMA supplied mechanism for mapping these return codes into a meaningful message (e.g., `IdmaSystem::GetResultCodeDescription`), until a DMA System object becomes available.

4.2.41.1.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_NETWORK_UNAVAILABLE	The network needed to perform this operation is not available.
DMARC_NOT_FOUND	Requested item not found.
DMARC_OK	(S_OK) Success.

4.2.41.2 IdmaSystemManager::EnumerateSystems

This method must be supported in every implementation of this interface.

4.2.41.2.1 Syntax

```
DmaRC IdmaSystemManager::EnumerateSystems (
    DMA_REFIID riid,
    pDmapv ppIEnumOfSystem)
```

4.2.41.2.2 Parameters

Name	Mode	Description
<i>riid</i>	input	Interface Identifier for the desired interface on resulting object.
<i>ppIEnumOfSystem</i>	output	Returns a reference to the requested interface or NULL if unsuccessful.

4.2.41.2.3 Description

The `IdmaSystemManager::EnumerateSystems` function returns an interface (typically `IdmaEnumOfObject`) to a COM object which the caller can use to enumerate DMA System objects which the System Manager was able to locate and connect to via `IdmaSystemManager::ConnectSystem` (using the

character set encoding id and locale name parameters specified in a prior call to `dmaConnectSystemManager`).

In general, the `IdmaAuthentication` interface must be exercised on the returned DMA System object before the object can be used for anything other than inspecting the System's non-object valued properties.

There is a small well defined set of DMA return codes returned from this method. There is no DMA supplied mechanism for mapping these return codes into a meaningful message (e.g., `IdmaSystem::GetResultCodeDescription`), until a DMA System object becomes available.

4.2.41.2.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.41.3 IdmaSystemManager::GetMalloc

This method must be supported in every implementation of this interface.

4.2.41.3.1 Syntax

DmaRC **IdmaSystemManager::GetMalloc** (
DMA_REFIID *riid*,
pDmapv *ppIMalloc*)

4.2.41.3.2 Parameters

Name	Mode	Description
<i>riid</i>	input	Interface Identifier for the desired interface on the system manager's Memory Management object.
<i>ppIMalloc</i>	output	Returns the selected interface of the COM Memory Management object used by this DMA System Manager and its children. This is an interface of the object whose interface was supplied as the <i>pIMalloc</i> parameter of the <code>dmaConnectSystemManager</code> operation if that parameter was non-NULL, otherwise it is an interface on the COM Memory Management object that the System Manager supplied. This is always the correct Memory Manager to use for freeing data referenced by non-interface output pointers returned from methods on this System Manager object or its children.

4.2.41.3.3 Description

Returns the selected interface for the memory manager used by this DMA System Manager object. This is the memory manager used for all allocations of data passed to the client as a result of a DMA method call. This interface must be used to free any of that data when the returned pointer is no longer needed.

See the Component Object Model specification for a complete definition of the IMalloc interface. The following specialized methods are included in this interface: Alloc, Free, Realloc, GetSize, DidAlloc, HeapMinimize. The interface specification does not require that all of these methods be supported by the object supplying the interface.

If the returned object does not support the interface specified by the riid parameter, the operation will fail and the result code DMARC_BAD_INTERFACE will be returned. In case the operation is successful, the returned interface must be released using the method IUnknown::Release.

4.2.41.3.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.42 IdmaTextOrdering

This is the interface delivered by text ordering service elements, providing string ordering using a particular text collation sequence.

4.2.42.1 IdmaTextOrdering::CompareStrings

This method must be supported in every implementation of this interface.

4.2.42.1.1 Syntax

```
DmaRC IdmaTextOrdering::CompareStrings (
    pDmaString pString1,
    pDmaString pString2,
    pDmaInteger32 pResult)
```


4.2.42.1.2 Parameters

Name	Mode	Description
<i>pString1</i>	input	Pointer to a DmaString
<i>pString2</i>	input	Pointer to a DmaString
<i>pResult</i>	output	Return value indicating whether string1 is greater than, equal or less than string2

4.2.42.1.3 Description

This method has semantics identical to the POSIX strcoll function in that it returns a result value of 0 if the strings are lexically identical, less than 0 if string1 is lexically less than string2 and greater than 0 if string1 is lexically greater than string2.

The CompareStrings method implements the lexical comparison per the conventions of the collation sequence offered by the text ordering service element. If the collation sequence is locale-dependent, then the locale with which the text ordering service object was instantiated will apply.

4.2.42.1.4 Return Values

Name	Description
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_OK	(S_OK) Success.

4.2.43 IdmaVersionable

This interface provides the ability to check in versions of versionable objects (initially Document Versions) in DMA.

4.2.43.1 IdmaVersionable::SetCheckIn

This method must be supported in every implementation of this interface.

4.2.43.1.1 Syntax

```
DmaRC IdmaVersionable::SetCheckIn (
    Dmapv pIVersionSeries,
    Dmapv pIVersionDescription)
```

4.2.43.1.2 Parameters

Name	Mode	Description
<i>pVersionSeries</i>	input	Pointer to an interface on a VersionSeries object
<i>pVersionDescription</i>	input	Pointer to an interface on a new Version Description object which points to this versionable object.

4.2.43.1.3 Description

Records the intention to perform a checkin of the versionable object against a specified version series. In order for this operation to succeed the client must be holding a reservation against the specified version series. In addition, if the Reservation object which records that reservation has a non-null value for the NewVersion property, then the current versionable object (the target of the SetCheckIn method) must be a scratchpad for that (cloned) object. This method call captures references to the version series and version description objects.

When the ExecuteChange operation is invoked following this method call, the following changes are made in the persistent store:

- The Versionable object is saved (if not already persistent) or updated. The VersionDescriptions property of the Versionable object will be extended to include the newly created version description (see below).
- The Reservation object resolved by the checkin is deleted.
- A new Version Description is created in the persistent store reflecting the extended properties of the referenced Version Description scratchpad at the time of ExecuteChange, and with appropriate generated values for the Version Series, Version and IsCurrentVersion properties.
- If there is a Current Version Description of the specified Version Series, its IsCurrentVersion property is updated to DMA_FALSE.
- The persistent state of the referenced Version Series is modified such that the Current Version Description property references the newly created Version Description and that the latter appears at the end of the Version Descriptions enum property.

The ExecuteChange operation is not permitted to change the scratchpads of the referenced Version Series or Version Description objects.

If the document space supports threaded versioning, then multiple calls to SetCheckIn may be allowed, even with the same parameters, without an intervening call on ExecuteChange. If the document space does not support

threaded versioning, this method fails, if the object has been previously checked in, or if there is already a checkin pending for this object

4.2.43.1.4 Return Values

Name	Description
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_CONFLICTING_OPERATION	The call on this primary intentional method logically conflicts with a prior call on a different primary intentional method.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_OK	(S_OK) Success.

4.2.43.1.5 Deferred Return Values

Name	Description
DMARC_ACCESS_DENIED	(E_ACCESSDENIED) The requester has insufficient access rights to perform the requested operation.
DMARC_MULTIPLE_CHECKIN_NOT_SUPPORTED	The implementation does not support multiple different checkins on the same object (i.e., threaded versioning).
DMARC_NO_RESERVATION	The Version Series is not holding a reservation at this time.
DMARC_OBJECT_LOCKED	The lock(s) on the persistent object prohibit the attempted operation.
DMARC_OBJECT_MODIFIED	The object has been modified since it was retrieved.

4.2.44 IdmaVersionSeries

This interface provides the ability to reserve and checkout versions of versionable objects (initially Document Versions) in DMA. It also provides a method to revoke (cancel) reservations and checkouts.

This interface is only present when the IdmaConnection interface is also present. The operations are specific to persistent objects and don't apply to any other forms of Version Series object.

4.2.44.1 IdmaVersionSeries::SetCheckOutNext

Support for this method is optional.

4.2.44.1.1 Syntax

DmaRC **IdmaVersionSeries::SetCheckOutNext** (
Dmapv *pReservation*)

4.2.44.1.2 Parameters

Name	Mode	Description
<i>pReservation</i>	input	Pointer to an interface of a new, unsaved reservation object

4.2.44.1.3 Description

Records the intention to reserve the right to check-in the next new current versionable object in this series.

Following ExecuteChange on the Version Series the persistent state of the Version Series is updated and a new Reservation object and a new working Versionable object are created in the persistent store.

The Reservation property of the persistent Version Series object will be updated to refer to the new Reservation object.

The Reserved For property of the persistent Reservation object will refer to the Version Series object, the New Version property will refer to the working Versionable object and any extended properties will be set to reflect the extended properties of the Reservation object at the time of the ExecuteChange call.

The working Versionable object is a clone of the current Versionable object in the version series. The Reservation and cloned versionable object can be found through navigation from the "Reservation" property of the Version Series

When the versionable object is "cloned", the versionable object and all dependent persistable subordinate objects are copied recursively. Cloning sets the navigation properties to null but preserves other properties with the exception of the OIID which is given a new value.

The major purpose in providing Checkout in addition to Reserve is to allow the Document Space to optimize the work associated with 'cloning' the Document Version, particularly the content data.

A reservation can be resolved with a check-in, or can be revoked. Revoking a reservation created by SetCheckOutNext does not delete the NewVersion object.

4.2.44.1.4 Return Values

Name	Description
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_CONFLICTING_OPERATION	The call on this primary intentional method logically conflicts with a prior call on a different primary intentional method.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_OK	(S_OK) Success.
DMARC_RESERVATION_PENDING	The connection is already holding a request for a reservation. There cannot be more than one at a time.

4.2.44.1.5 Deferred Return Values

Name	Description
DMARC_NO_CURRENT_VERSION	The version series has no current version.
DMARC_OBJECT_LOCKED	The lock(s) on the persistent object prohibit the attempted operation.
DMARC_OBJECT_MODIFIED	The object has been modified since it was retrieved.
DMARC_RESERVATION_EXISTS	The version series is already holding a reservation.
DMARC_RESERVATION_NOT_ALLOWED	The condition of the version series does not permit a reservation, though no reservation is currently present.

4.2.44.2 IdmaVersionSeries::SetReserveNext

This method must be supported in every implementation of this interface.

4.2.44.2.1 Syntax

DmaRC **IdmaVersionSeries::SetReserveNext** (
Dmapv *pReservation*)

4.2.44.2.2 Parameters

Name	Mode	Description
<i>pReservation</i>	input	Pointer to an interface of a new, unsaved reservation object

4.2.44.2.3 Description

Records the intention to reserve the right to check-in the next new current versionable object in this series.

Following `ExecuteChange` on the Version Series the persistent state of the Version Series is updated and a new Reservation object is created in the persistent store.

The Reservation property of the persistent Version Series object will be updated to refer to the new Reservation object.

The Reserved For property of the persistent Reservation object will refer to the Version Series object, the New Version property (if supported) will be NULL, and any extended properties will be set to reflect the extended properties of the Reservation object at the time of the `ExecuteChange` call.

Unlike `SetCheckOutNext`, `SetReserveNext` does not create a clone of the versionable object.

A reservation can be resolved with a check-in, or can be revoked.

4.2.44.2.4 Return Values

Name	Description
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_CONFLICTING_OPERATION	The call on this primary intentional method logically conflicts with a prior call on a different primary intentional method.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_OK	(S_OK) Success.
DMARC_RESERVATION_PENDING	The connection is already holding a request for a reservation. There cannot be more than one at a time.

4.2.44.2.5 Deferred Return Values

Name	Description
DMARC_ACCESS_DENIED	(E_ACCESSDENIED) The requester has insufficient access rights to perform the requested operation.
DMARC_OBJECT_LOCKED	The lock(s) on the persistent object prohibit the attempted operation.
DMARC_OBJECT_MODIFIED	The object has been modified since it was retrieved.
DMARC_RESERVATION_EXISTS	The version series is already holding a reservation.
DMARC_RESERVATION_NOT_ALLOWED	The condition of the version series does not permit a reservation, though no reservation is currently present.

4.2.44.3 `IdmaVersionSeries::SetRevoke`

This method must be supported in every implementation of this interface.

4.2.44.3.1 Syntax

DmaRC IdmaVersionSeries::SetRevoke ()

4.2.44.3.2 Description

Records the intention to cancel an existing CheckOut or Reservation for the Version Series object. On successful completion of this method and one of the ExecuteChange(s) methods the Reservation property of the series is set to NULL and the Reservation object formerly pointed to is deleted.

If the Reservation was created by SetCheckOutNext, the cloned NewVersion object (created as a result of that method call) is NOT deleted. If the NewVersion should be deleted, then the application must explicitly delete the object (which is pointed to by the Reservation's NewVersion property).

This method will not fail because there is a SetRevoke already pending.

4.2.44.3.3 Return Values

Name	Description
DMARC_CONFLICTING_OPERATION	The call on this primary intentional method logically conflicts with a prior call on a different primary intentional method.
DMARC_OK	(S_OK) Success.

4.2.44.3.4 Deferred Return Values

Name	Description
DMARC_NO_RESERVATION	The Version Series is not holding a reservation at this time.

4.2.45 IUnknown

Microsoft COM IUnknown interface. Included here for completeness, with arguments given as equivalent DMA types.

4.2.45.1 IUnknown::AddRef

This method must be supported in every implementation of this interface.

4.2.45.1.1 Syntax

DmaUInteger32 IUnknown::AddRef ()

4.2.45.1.2 Description

Called when another component object is using the interface. Increases the object's reference count by one and returns the number of references to the

object. As long as the reference count is non-zero, the object must remain in memory.

4.2.45.2 IUnknown::QueryInterface

This method must be supported in every implementation of this interface.

4.2.45.2.1 Syntax

```
DmaRC IUnknown::QueryInterface (
    DMA_REFIID riid,
    pDmapv ppIObject)
```

4.2.45.2.2 Parameters

Name	Mode	Description
<i>riid</i>	input	Interface Identifier for the desired interface.
<i>ppIObject</i>	output	Location for the returned interface pointer.

4.2.45.2.3 Description

This method requests an interface pointer from a COM object. If the interface is supported by the object, this method returns DMARC_OK and places the interface pointer in the location indicated by ppIObject. If the interface is not supported, the error code DMARC_BAD_INTERFACE is returned.

This is the mechanism in COM that allows clients to dynamically discover (at run time) whether or not an interface is supported by a component object.

If an error occurs, NULL is returned as the new interface pointer.

4.2.45.2.4 Return Values

Name	Description
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_OK	(S_OK) Success.
DMARC_UNEXPECTED	(E_UNEXPECTED) An unexpected error occurred.

4.2.45.3 IUnknown::Release

This method must be supported in every implementation of this interface.

4.2.45.3.1 Syntax

```
DmaUInteger32 IUnknown::Release ( )
```


4.2.45.3.2 Description

Called when the client no longer requires use of the interface. Decrements the reference count and returns the number of references to the object. This method is used to free interface pointers returned to the client interface. When all interface pointers to an object are released, it will be deleted.

This method must not be used for freeing references to other datatypes (e.g., DMAString or Binary buffers).

4.3 Macros

This section describes the macros defined for the DMA specification.

4.3.1 DMA_CREATE_STRING

4.3.1.1 Syntax

```
void DMA_CREATE_STRING (  
void* _allocfn(size_t),  
const DmaUInteger32 _charcount,  
const DmaWChar * _srcwchars,  
ppDmaString _ppstr)
```

4.3.1.2 Parameters

Name	Mode	Description
<i>_allocfn(size_t)</i>	input	A pointer to allocator function.
<i>_charcount</i>	input	The number of DmaWChar characters supplied in <i>srcwchars</i> .
<i>_srcwchars</i>	input	The source character string for initializing the DmaString.
<i>_ppstr</i>	output	pointer to where the pDmaString of the new string is delivered

4.3.1.3 Description

This is a macro. Clients should assume this macro (which is a little intricate) is a block of code. Clients should not assume it is a function or expression, even though its behavior is as if it is implemented by a procedure.

DMA_CREATE_STRING creates a new DmaString using the supplied allocator. Because DMA_CREATE_STRING is a macro, the supplied allocator can be a method on an interface (e.g., *imalloc->Alloc*). The initial value of the string is a copy of the first *_charcount* characters of *_srcwchars*. The pointer to the constructed DmaString is delivered into the output location, *_ppstr*.

The input arguments must be valid. The macros are NOT expected to check for valid input arguments. The macro will either succeed or return with **_ppstr* set to NULL.

4.3.2 DMA_FREE_STRING

4.3.2.1 Syntax

```
void DMA_FREE_STRING (
    void _freeFn( void *memblock ),
    pDmaString _pstr)
```

4.3.2.2 Parameters

Name	Mode	Description
<i>_freeFn(void *memblock)</i>	input	The pointer to the function to be used for freeing the memory
<i>_pstr</i>	input	The pointer to the DmaString to be freed

4.3.2.3 Description

This is a macro. Clients should assume this macro is a block of code. Clients should not assume it is a function or expression.

The macro free's the DmaString using the specified de-allocator. It is legal to supply a NULL *_pstr* value, in which case the macro perform no operation.

There are no error returns.

4.3.3 DMA_GET_STRING_CHAR_COUNT

4.3.3.1 Syntax

```
DmaUInteger32 DMA_GET_STRING_CHAR_COUNT (
    const pDmaString _pstr)
```

4.3.3.2 Parameters

Name	Mode	Description
<i>_pstr</i>	input	The DmaString

4.3.3.3 Description

This is a macro. Clients can safely assume this macro is an expression.

The macro returns the number of characters in the string. The terminating NULL of a string is not counted.

There are no error returns. The macro returns 0 if a NULL input is supplied.

4.3.4 DMA_GET_STRING_TEXT

4.3.4.1 Syntax

```
const DmaWChar* DMA_GET_STRING_TEXT (  
    pDmaString _pstr)
```

4.3.4.2 Parameters

Name	Mode	Description
<i>_pstr</i>	input	The DmaString

4.3.4.3 Description

This is a macro. Clients can safely assume this macro is an expression.

The macro returns a pointer to a sequence of characters. There is a null character at the end.

There are no error returns. The macro returns NULL, if a NULL input is supplied.

4.4 DMA Return Codes

All DMA methods return an integer return code following the conventions used for an HRESULT in the COM specification. This table lists symbolic names and descriptions for the DMA return codes, all of which start with “DMARC_”.

For background on HRESULT, see <http://www.microsoft.com/oledev/olecom/Ch03.htm#IUnknown>

Return code listings in the method descriptions are only partial. Other generic codes (such as DMARC_FAILED) can occur. Some codes are warnings instead of errors – the descriptions of these warning codes begin with “(Warning)”.

Name	Description
DMARC_3VE_NOT_SUPPORTED	The current scope does not support three valued elimination.
DMARC_ABORT	The progress callback indicated that the method should be aborted.
DMARC_ACCESS_DENIED	(E_ACCESSDENIED) The requester has insufficient access rights to perform the requested operation.
DMARC_ALIAS_CONFLICT	A class, operator or property in one component scope unifies with more than one such entity in another component scope.
DMARC_ALREADY_AUTHENTICATED	The user is already authenticated.
DMARC_AT_ENUM_END	The enumerator object is positioned at the end of the component objects.
DMARC_BAD_CLASSID	The supplied identifier does not reference an available class of object.
DMARC_BAD_COLLATION_SEQUENCE	The specified collation sequence is not known or supported by the scope.
DMARC_BAD_DOC_SPACE	There is no available document space with the specified Dmald
DMARC_BAD_FROM_EXPRESSION	The query contains an invalid From Expression.
DMARC_BAD_INDEX	The index specified does not reference a valid property or list element.
DMARC_BAD_INTERFACE	(E_NOINTERFACE) The requested interface is not supported by this object.
DMARC_BAD_LOCK_TYPE	The lock type is invalid.
DMARC_BAD_OBJECT	An inappropriate object was passed as parameter to this function. For example, the object does not support a required interface or is of an inappropriate class.
DMARC_BAD_OIID	The supplied buffer does not contain a valid OIID.
DMARC_BAD_OPERAND	An operand is incorrect in datatype, form, or class.

Name	Description
DMARC_BAD_OPERATOR	An operator ID is not defined in the metadata of the current scope.
DMARC_BAD_ORDERBY_ELEMENT	Illegal 'order by' list element.
DMARC_BAD_PARAMETER	(E_INVALIDARG) Invalid input parameter.
DMARC_BAD_PROPID	A property identifier is not valid.
DMARC_BAD_PROTECTION_LEVEL	Illegal value for the protection level parameter.
DMARC_BAD_RC	Invalid Result Code.
DMARC_BAD_REFERENCE	The object cannot be saved because it references another object which is not persistent.
DMARC_BAD_RESERVATION	One or more of the Reservations is not the current reservation in the VersionSeries to which it refers.
DMARC_BAD_SC_OCCUR	The value of a query node's searchable class occurrence property is not present in the query.
DMARC_BAD_SELECT_ELEMENT	Illegal select list element.
DMARC_BAD_SUBQUERY_DATATYPE	The datatype of the query root node of an "IN" subquery does not match that of the property in the select list.
DMARC_BAD_SUBQUERY_SELECT	The select list of a subquery contains an inappropriate number of elements
DMARC_BAD_URL	The resource URL is not syntactically valid.
DMARC_BAD_VALUE	The value given for a property or list element lies outside the permitted range or value set, or exceeds the maximum length allowed.
DMARC_CLASS_NOT_SEARCHABLE	A class referred to in the From Expression is not searchable.
DMARC_CLASSES_NOT_JOINABLE	A particular join operator in the query is supported, but the operator does not support the join between the particular operands specified in the query.
DMARC_CONFLICTING_OPERATION	The call on this primary intentional method logically conflicts with a prior call on a different primary intentional method.
DMARC_CONSTRAINT_VIOLATED	The operation violates a constraint of the implementation.
DMARC_DATATYPE_MISMATCH	The method invoked is inappropriate for the datatype of the property.
DMARC_DEVICE_ERROR	An error has occurred reading or writing a hardware device.
DMARC_DISCONNECTED	The logical connection to the document space has been disconnected.
DMARC_DISTINCT_NOT_SUPPORTED	The scope does not support the request for distinct result rows.
DMARC_EXTRA_OPERANDS	A query node has extra operands.
DMARC_FAILED	(E_FAIL) The operation failed.

Name	Description
DMARC_FOREIGN_OBJECT	An object is not from the current document space.
DMARC_ILLEGAL_OPERATION	The operation is disallowed by the DMA Specification.
DMARC_LIST_BOUNDS_ERROR	The number of elements in a list is outside the permitted range.
DMARC_LOST_CONNECTION	The logical connection of the current object to the persistent store has been lost permanently. The operation could not be completed.
DMARC_MAX_ROWS	A result set was truncated because it exceeded the maximum result items parameter.
DMARC_MISSING_OPERANDS	A query node is missing one or more operands.
DMARC_MISSING_REFERENCE	The object cannot be saved because required reference to it is absent. For example, it may be required to be in a container and no appropriate relationship object exists.
DMARC_MULTIPLE_CHECKIN_NOT_SUPPORTED	The implementation does not support multiple different checkins on the same object (i.e., threaded versioning).
DMARC_NEED_MORE_DATA	(Warning) Indicates that more data is required to complete the browse connection operation.
DMARC_NETWORK_ERROR	A network error has occurred. It is undetermined as to whether retrying the method might succeed. The operation could not be completed.
DMARC_NETWORK_UNAVAILABLE	The network needed to perform this operation is not available.
DMARC_NO_COLLATIONS	Merged scope creation failed because there would be no supported collation sequences.
DMARC_NO_CURRENT_VERSION	The version series has no current version.
DMARC_NO_MEMORY	(E_OUTOFMEMORY) Insufficient memory to complete the operation.
DMARC_NO_RESERVATION	The Version Series is not holding a reservation at this time.
DMARC_NOT_AUTHENTICATED	The user is not authenticated.
DMARC_NOT_CREATABLE	Creation of objects of the specified class is not supported.
DMARC_NOT_FOUND	Requested item not found.
DMARC_NOT_LOCKED	The object is not locked.
DMARC_NOT_PUBLISHED	The service registry entry is not currently published.
DMARC_NOT_SUPPORTED	This method is not supported in the context of this session or object.
DMARC_NOT_UNIQUE	A uniqueness requirement has been violated.
DMARC_OBJECT_DELETED	The object has been deleted since it was retrieved.
DMARC_OBJECT_LOCKED	The lock(s) on the persistent object prohibit the attempted operation.

Name	Description
DMARC_OBJECT_MODIFIED	The object has been modified since it was retrieved.
DMARC_OBJECT_REFERENCED	The object cannot be deleted because it is referenced by other objects.
DMARC_OK	(S_OK) Success.
DMARC_OPERAND_MERGE_CONFLICT	The merge would create an invalid operand description.
DMARC_OPERATOR_MERGE_CONFLICT	An operator could not be merged because it has conflicting signatures in component scopes.
DMARC_POSITION_NOT_ALLOWED	The stream cannot be positioned as requested.
DMARC_PROPERTY_MERGE_CONFLICT	A property could not be merged because there is a conflict of conflicting datatype or cardinality in component scopes.
DMARC_PROPERTY_NOT_ORDERABLE	A property referred to in the query order by list is not orderable.
DMARC_PROPERTY_NOT_SEARCHABLE	A property referred to in the Query Expression is not searchable.
DMARC_PROPERTY_NOT_SELECTABLE	A property referred to in the Selections list is not selectable.
DMARC_PROTECTION_LEVEL_NOT_SUPPORTED	The desired protection level is not supported by the doc space.
DMARC_QUERY_AT_END	The query ran to successful completion without encountering any constraints on time limits, number of rows generated, or any other constraints, and there are no more result rows that satisfy the query. A result row is not returned. No more result rows will be returned unless ReExecuteQuery is executed.
DMARC_QUERY_TREE_LOOP	There is a circular reference loop or a shared node in the query.
DMARC_READ_ONLY	Method failed because an object or property is read-only.
DMARC_REFERENCES_OTHERS	The object cannot be deleted because it holds references to other objects.
DMARC_REQUIRED_VALUE_ABSENT	A required property value has not been set.
DMARC_RESERVATION_EXISTS	The version series is already holding a reservation.
DMARC_RESERVATION_NOT_ALLOWED	The condition of the version series does not permit a reservation, though no reservation is currently present.
DMARC_RESERVATION_PENDING	The connection is already holding a request for a reservation. There cannot be more than one at a time.
DMARC_RESOURCE_NOT_FOUND	The indicated resource was not found.
DMARC_RESULTS_TRUNCATED	The result set was truncated explicitly by a call on IdmaResultSet:: TerminateResults. No further rows will be returned unless ReExecuteQuery is executed. This is a failure code - no row is returned.
DMARC_SHORT_READ	(Warning) End of stream was encountered before the requested number of bytes was read.

Name	Description
DMARC_STALE_METADATA	The metadata in the persistent store has been changed so that it does not match the client's current copy of the metadata.
DMARC_TIMEOUT	A time limit has been exceeded.
DMARC_TRUNCATED	Result sets were truncated in different component scopes for different reasons.
DMARC_UNEXPECTED	(E_UNEXPECTED) An unexpected error occurred.
DMARC_UNSUPPORTED_ORDERING	An unacceptable value was provided for an ordering parameter.
DMARC_URL_PROTOCOL_NOT_SUPPORTED	The URL protocol specified in the reference is not supported.
DMARC_VALUE_NOT_SET	The requested property currently has no value.

4.5 Join Operators

A join operator node has two or three operands:

- Operand 0 may be a searchable class node or a join operator node
- Operand 1 is required to be a searchable class node

For a cross join, these are the only operands permitted. For all the other join types, a third operand is required:

- Operand 2 must be a normal query operator node for which the value of Join Participation is DMA_JOIN_PARTICIPATION_OPERAND, e.g. an “equal” operator.
- The operands of the operand 2 node must be properties of searchable classes. One of the operands of the operand 2 node must be a property from the searchable class of operand 1. The other must be from one of the searchable classes in the subtree of operand 0.

The value of the Operand Datatype property of the Query Operand Description of operands 0 and 1 of join operators is DMA_DATATYPE_CLASS .

For cross joins, the Cartesian product is produced. For all other join operators, the searchable class or subordinate join expression from operand 0 is joined to the searchable class of operand 1 via the Boolean expression subtree whose root is operand 2.

It is not required that a Join Operator be supported between all possible pairs of searchable classes in the document space.

The well known join operators defined by DMA do not join objects from different document spaces. When a join operation is actually performed, it is performed by a single document space between searchable classes in its metadata. Undefined parts of the From Expression must be eliminated by the rules of three valued elimination. If three valued elimination is needed but is not available, the query fails with a query construction error. For the well known join operators specified by DMA, the results of evaluating a join in a merged scope is the union of the results of evaluating the join in its component scopes.

4.5.1 Joins and Three Valued Elimination

In the following tables, operand 0 is undefined if it is a searchable class node which specifies an undefined class, or if it is a join operator node which evaluates to undefined by recursive application of these rules.

Operand 1 is “Undefined” if it is a searchable class node which specifies an undefined class.

Operand 2 is “Undefined” if it specifies an operator which is undefined, or if it evaluates to “Undefined” because of “Undefined” operands.

An outcome of "Undefined" for the root node of the From Expression results in the query returning zero rows.

4.5.2 Cross Join

Op 0	Op 1	Outcome
Defined	Undefined	Undefined
Undefined	Defined	Undefined
Undefined	Undefined	Undefined
Defined	Defined	Defined

4.5.3 Inner Join

Op 0	Op 1	Op 2	Outcome
Defined	Defined	Undefined	Undefined
Defined	Undefined	Defined	Undefined
Defined	Undefined	Undefined	Undefined
Undefined	Defined	Defined	Undefined
Undefined	Defined	Undefined	Undefined
Undefined	Undefined	Defined	Undefined
Undefined	Undefined	Undefined	Undefined
Defined	Defined	Defined	Defined

4.5.4 Left Outer Join

Op 0	Op 1	Op 2	Outcome
Defined	Defined	Undefined	Undefined
Defined	Undefined	Defined	Defined (result is Op 0)
Defined	Undefined	Undefined	Defined (result is Op 0)
Undefined	Defined	Defined	Undefined
Undefined	Defined	Undefined	Undefined
Undefined	Undefined	Defined	Undefined
Undefined	Undefined	Undefined	Undefined
Defined	Defined	Defined	Defined

4.5.5 Right Outer Join

Op 0	Op 1	Op 2	Outcome
Defined	Defined	Undefined	Undefined
Defined	Undefined	Defined	Undefined
Defined	Undefined	Undefined	Undefined
Undefined	Defined	Defined	Defined (result is Op 1)
Undefined	Defined	Undefined	Defined (result is Op 1)
Undefined	Undefined	Defined	Undefined

Op 0	Op 1	Op 2	Outcome
Undefined	Undefined	Undefined	Undefined
Defined	Defined	Defined	Defined

4.5.6 DMA Defined Join Operators

The following Join Operators are defined by DMA 1.0:

Name	Define Name	Description
Cross Join	dmaJoinOperator_Cross	Returns the Cartesian product of the rows of the two searchable classes.
Inner Join	dmaJoinOperator_Inner	A Join operation that does not return a row if either the left or right or both operands of the equality operator are NULL for a particular row under scan.
Left Outer Join	dmaJoinOperator_LeftOuter	Returns a row if the left operand of the equality operator is non NULL, even if the right operand is NULL.
Right Outer Join	dmaJoinOperator_RightOuter	Returns a row if the right operand of the equality operator is non NULL, even if the left operand is NULL.

4.6 DMA Well Known Query Operators

4.6.1 Ordering Rules

For all the ordering relational operators (greater, greater or equal, less, less or equal), for all datatypes, the null value collates before any non null value.

Two strings of unequal length are always unequal. If two strings are of unequal length, and the initial prefix of the longer string is equal to the entire shorter string, then the shorter string collates before the longer string. The collating sequence used in the comparison for all the string relational operators is specified by a property on the Query Object. Refer to the discussion on the query object.

The comparison of binary operands is done one unsigned byte at a time from lower index values to higher index values. Two binary operands of unequal length are always unequal. If the two binary operands are of unequal length, and the initial prefix of the longer operand is equal to the entire shorter operand, then the shorter operand collates before the longer operand.

The comparison of lists is done one element at a time, starting from element zero. The first corresponding pair of list elements with unequal values determines the outcome of the compare. If no corresponding pair of list elements have unequal values, the lists are considered equal. Two lists of unequal lengths are always unequal. If two lists are of unequal length, and the initial elements of the longer list are equal to the corresponding elements in the shorter list, then the shorter list collates before the longer list.

Note: The above rules are in harmony with the ANSI/ISO SQL 92 standards: ANSI X3.135-1992, "Database Language SQL", and ISO/IEC 9075:1992, "Database Language SQL".

When a singleton is compared against a list, the singleton is treated as if it were a list of one element, and the comparison proceeds as for two lists.

4.6.2 DMA Well Known Query Operators Descriptions

Name	Result Type	Min # of Opnds	Max # of Opnds	Operand Type(s)	Safe to Eliminate	Define
And	Boolean	2	-1	Boolean	No	dmaQueryOperator_And
Or	Boolean	2	-1	Boolean	No	dmaQueryOperator_Or
Not	Boolean	1	1	Boolean	Yes	dmaQueryOperator_Not
Equal Binary	Boolean	2	2	Binary	Yes	dmaQueryOperator_EqualBinary
Unequal Binary	Boolean	2	2	Binary	Yes	dmaQueryOperator_UnequalBinary

Name	Result Type	Min # of Opnds	Max # of Opnds	Operand Type(s)	Safe to Eliminate	Define
Greater Binary	Boolean	2	2	Binary	Yes	dmaQueryOperator_GreaterBinary
Greater Or Equal Binary	Boolean	2	2	Binary	Yes	dmaQueryOperator_GreaterOrEqualBinary
Less Binary	Boolean	2	2	Binary	Yes	dmaQueryOperator_LessBinary
Less Or Equal Binary	Boolean	2	2	Binary	Yes	dmaQueryOperator_LessOrEqualBinary
Equal String	Boolean	2	2	String	Yes	dmaQueryOperator_EqualString
Unequal String	Boolean	2	2	String	Yes	dmaQueryOperator_UnequalString
Greater String	Boolean	2	2	String	Yes	dmaQueryOperator_GreaterString
Greater Or Equal String	Boolean	2	2	String	Yes	dmaQueryOperator_GreaterOrEqualString
Less String	Boolean	2	2	String	Yes	dmaQueryOperator_LessString
Less Or Equal String	Boolean	2	2	String	Yes	dmaQueryOperator_LessOrEqualString
Equal Boolean	Boolean	2	2	Boolean	Yes	dmaQueryOperator_EqualBoolean
Unequal Boolean	Boolean	2	2	Boolean	Yes	dmaQueryOperator_UnequalBoolean
Equal Integer32	Boolean	2	2	Int32	Yes	dmaQueryOperator_EqualInteger32
Unequal Integer32	Boolean	2	2	Int32	Yes	dmaQueryOperator_UnequalInteger32
Greater Integer32	Boolean	2	2	Int32	Yes	dmaQueryOperator_GreaterInteger32
Greater Or Equal Integer32	Boolean	2	2	Int32	Yes	dmaQueryOperator_GreaterOrEqualInteger32
Less Integer32	Boolean	2	2	Int32	Yes	dmaQueryOperator_LessInteger32
Less Or Equal Integer32	Boolean	2	2	Int32	Yes	dmaQueryOperator_LessOrEqualInteger32
Equal Float64	Boolean	2	2	Float64	Yes	dmaQueryOperator_EqualFloat64
Unequal Float64	Boolean	2	2	Float64	Yes	dmaQueryOperator_UnequalFloat64
Greater Float64	Boolean	2	2	Float64	Yes	dmaQueryOperator_GreaterFloat64

Name	Result Type	Min # of Opnds	Max # of Opnds	Operand Type(s)	Safe to Eliminate	Define
Greater Or Equal Float64	Boolean	2	2	Float64	Yes	dmaQueryOperator_GreaterOrEqualFloat64
Less Float64	Boolean	2	2	Float64	Yes	dmaQueryOperator_LessFloat64
Less Or Equal Float64	Boolean	2	2	Float64	Yes	dmaQueryOperator_LessOrEqualFloat64
Equal Date Time	Boolean	2	2	DateTime	Yes	dmaQueryOperator_EqualDateTime
Unequal Date Time	Boolean	2	2	DateTime	Yes	dmaQueryOperator_UnequalDateTime
Greater Date Time	Boolean	2	2	DateTime	Yes	dmaQueryOperator_GreaterDateTime
Greater Or Equal Date Time	Boolean	2	2	DateTime	Yes	dmaQueryOperator_GreaterOrEqualDateTime
Less DateTime	Boolean	2	2	DateTime	Yes	dmaQueryOperator_LessDateTime
Less Or Equal Date Time	Boolean	2	2	DateTime	Yes	dmaQueryOperator_LessOrEqualDateTime
Equal Id	Boolean	2	2	ID	Yes	dmaQueryOperator_EqualId
Unequal Id	Boolean	2	2	ID	Yes	dmaQueryOperator_UnequalId
Equal Object	Boolean	2	2	Object	Yes	dmaQueryOperator_EqualObject
Is Defined	Boolean	1	1	Any	No	dmaQueryOperator_IsDefined
Is Null	Boolean	1	1	Any	Yes	dmaQueryOperator_IsNull
Like	Boolean	3	3	String	Yes	dmaQueryOperator_Like
Is Class	Boolean	2	2	Int32; ID	Yes	dmaQueryOperator_IsClass
Add Integer32	Int32	2	-1	Int32	Yes	dmaQueryOperator_AddInteger32
Subtract Integer32	Int32	2	2	Int32	Yes	dmaQueryOperator_SubtractInteger32
Negate Integer32	Int32	1	1	Int32	Yes	dmaQueryOperator_NegateInteger32
Absolute Value Integer32	Int32	1	1	Int32	Yes	dmaQueryOperator_AbsoluteValueInteger32
Multiply Integer32	Int32	2	-1	Int32	Yes	dmaQueryOperator_MultiplyInteger32
Divide Integer32	Int32	2	2	Int32	Yes	dmaQueryOperator_DivideInteger32
Add Float64	Float64	2	-1	Float64	Yes	dmaQueryOperator_AddFloat64

Name	Result Type	Min # of Opnds	Max # of Opnds	Operand Type(s)	Safe to Eliminate	Define
Subtract Float64	Float64	2	2	Float64	Yes	dmaQueryOperator_SubtractFloat64
Negate Float64	Float64	1	1	Float64	Yes	dmaQueryOperator_NegateFloat64
Absolute Value Float64	Float64	1	1	Float64	Yes	dmaQueryOperator_AbsoluteValueFloat64
Multiply Float64	Float64	2	-1	Float64	Yes	dmaQueryOperator_MultiplyFloat64
Divide Float64	Float64	2	2	Float64	Yes	dmaQueryOperator_DivideFloat64
Integer32 To Float64	Float64	1	1	Int32	Yes	dmaQueryOperator_Integer32ToFloat32
Float64 To Integer32 Round	Int32	1	1	Float64	Yes	dmaQueryOperator_Float64ToInteger32Round
Float64 To Integer32 Truncate	Int32	1	1	Float64	Yes	dmaQueryOperator_Float64ToInteger32Truncate
Exists	Boolean	1	1	Class	Yes	dmaQueryOperator_Exists
In Binary	Boolean	2	2	Binary	Yes	dmaQueryOperator_InBinary
In Boolean	Boolean	2	2	Boolean	Yes	dmaQueryOperator_InBoolean
In String	Boolean	2	2	String	Yes	dmaQueryOperator_InString
In Integer32	Boolean	2	2	Int32	Yes	dmaQueryOperator_InInteger32
In Float64	Boolean	2	2	Float64	Yes	dmaQueryOperator_InFloat64
In Date Time	Boolean	2	2	DateTime	Yes	dmaQueryOperator_InDateTime
In Id	Boolean	2	2	ID	Yes	dmaQueryOperator_InId

4.6.2.1 dmaQueryOperator_And

Computes the logical AND of its Boolean operands. The AND operator must allow any operand to be constant.

4.6.2.2 dmaQueryOperator_Or

Computes the logical inclusive OR of its Boolean operands. The OR operator must allow any operand to be constant.

4.6.2.3 dmaQueryOperator_Not

Computes the logical negation of its Boolean operand

4.6.2.4 dmaQueryOperator_EqualBinary

Returns True if and only if the two Binary operands are equal

4.6.2.5 dmaQueryOperator_UnequalBinary

Returns True if and only if the two Binary operands are unequal

4.6.2.6 dmaQueryOperator_GreaterBinary

Returns True if and only if the first operand is greater than the second

4.6.2.7 dmaQueryOperator_GreaterOrEqualBinary

Returns True if and only if the first operand is greater than or equal to the second

4.6.2.8 dmaQueryOperator_LessBinary

Returns True if and only if the first operand is less than the second

4.6.2.9 dmaQueryOperator_LessOrEqualBinary

Returns True if and only if the first is less than or equal to the second

4.6.2.10 dmaQueryOperator_EqualString

Returns True if and only if the two string operands are equal

4.6.2.11 dmaQueryOperator_UnequalString

Returns True if and only if the two string operands are unequal

4.6.2.12 dmaQueryOperator_GreaterString

Returns True if and only if the first operand is greater than the second

4.6.2.13 dmaQueryOperator_GreaterOrEqualString

Returns True if and only if the first operand is greater than or equal to the second

4.6.2.14 dmaQueryOperator_LessString

Returns True if and only if the first operand is less than the second

4.6.2.15 dmaQueryOperator_LessOrEqualString

Returns True if and only if the first is less than or equal to the second

4.6.2.16 dmaQueryOperator_EqualBoolean

Returns True if and only if the two boolean operands are equal

4.6.2.17 dmaQueryOperator_UnequalBoolean

Returns True if and only if the two boolean operands are unequal

4.6.2.18 dmaQueryOperator_EqualInteger32

Returns True if and only if the two integer operands are equal

4.6.2.19 dmaQueryOperator_UnequalInteger32

Returns True if and only if the two integer operands are unequal

4.6.2.20 dmaQueryOperator_GreaterInteger32

Returns True if and only if the first operand is greater than the second

4.6.2.21 dmaQueryOperator_GreaterOrEqualInteger32

Returns True if and only if the first operand is greater than or equal to the second

4.6.2.22 dmaQueryOperator_LessInteger32

Returns True if and only if the first operand is less than the second

4.6.2.23 dmaQueryOperator_LessOrEqualInteger32

Returns True if and only if the first is less than or equal to the second

4.6.2.24 dmaQueryOperator_EqualFloat64

Returns True if and only if the first operand is equal to the second

4.6.2.25 dmaQueryOperator_UnequalFloat64

Returns True if and only if the two operands are unequal

4.6.2.26 dmaQueryOperator_GreaterFloat64

Returns True if and only if the first operand is greater than the second

4.6.2.27 dmaQueryOperator_GreaterOrEqualFloat64

Returns True if and only if the first operand is greater than or equal to the second

4.6.2.28 dmaQueryOperator_LessFloat64

Returns True if and only if the first operand is less than the second

4.6.2.29 dmaQueryOperator_LessOrEqualFloat64

Returns True if and only if the first is less than or equal to the second

4.6.2.30 dmaQueryOperator_EqualDateTime

Returns True if and only if the first operand is equal to the second

4.6.2.31 dmaQueryOperator_UnequalDateTime

Returns True if and only if the two operands are unequal

4.6.2.32 dmaQueryOperator_GreaterDateTime

Returns True if and only if the first operand is greater than the second

4.6.2.33 dmaQueryOperator_GreaterOrEqualDateTime

Returns True if and only if the first operand is greater than or equal to the second

4.6.2.34 dmaQueryOperator_LessDateTime

Returns True if and only if the first operand is less than the second

4.6.2.35 dmaQueryOperator_LessOrEqualDateTime

Returns True if and only if the first is less than or equal to the second

4.6.2.36 dmaQueryOperator_EqualId

Returns True if and only if the first operand is equal to the second

4.6.2.37 dmaQueryOperator_UnequalId

Returns True if and only if the two operands are unequal

4.6.2.38 dmaQueryOperator_EqualObject

This operator evaluates to DMA_TRUE if and only if the two operands are the same persistent object. This operator can be used, for example, for equi-joins on single-object-valued navigation properties, e.g., dmaProp_This, dmaProp_Head, dmaProp_Tail, etc.

4.6.2.39 dmaQueryOperator_IsDefined

The operand must be a property of a persistent object, not a constant or query parameter or operator or etc. Return True if the property is defined in the current scope, else False. It is used to help refine searches across multiple repositories when the rules of three valued logic are not sufficient to achieve the desired result

4.6.2.40 dmaQueryOperator_IsNull

The operand must be a property of a persistent object, not a constant or query parameter or operator or etc. The property must be defined in the current scope, or else DMARC_UNKNOWN is returned. If the property is defined, the operator returns True if the value of the property is NULL, else False.

4.6.2.41 dmaQueryOperator_Like

The first operand is the string under examination. The second operand is the pattern string, which may contain wildcard characters. The wildcard character "%" represents zero or more characters. The wildcard character "_" (underscore) represents exactly one character. The third operand is a single character string designating the escape character used in the pattern string. Return True if the pattern match is successful, else False. If the first operand is not defined or is NULL, return DMARC_UNKNOWN

4.6.2.42 dmaQueryOperator_IsClass

This operator can be used to refine 'include subclasses' searches so that fewer than all subclasses of the ancestor class are returned by the query. The first parameter is the searchable class occurrence number of a searchable class in the current query. This searchable class occurrence must have dmaProp_IncludeSubclassesRequested set to True. The second parameter is the ID of the searchable class or one of its subclasses. Return True if the searchable class ID of the current row under scan corresponding to the designated searchable class occurrence is equal to the second operand, else False.

4.6.2.43 dmaQueryOperator_AddInteger32

Return the sum of the operands

4.6.2.44 dmaQueryOperator_SubtractInteger32

Subtract the second operand from the first and return the difference

4.6.2.45 dmaQueryOperator_NegateInteger32

Return the negation of the operand

4.6.2.46 dmaQueryOperator_AbsoluteValueInteger32

Return the absolute value of the operand

4.6.2.47 dmaQueryOperator_MultiplyInteger32

Return the product of the operands

4.6.2.48 dmaQueryOperator_DivideInteger32

Divide the first operand by the second and return the quotient

4.6.2.49 dmaQueryOperator_AddFloat64

Return the sum of the operands

4.6.2.50 dmaQueryOperator_SubtractFloat64

Subtract the second operand from the first and return the difference

4.6.2.51 dmaQueryOperator_NegateFloat64

Return the negation of the operand

4.6.2.52 dmaQueryOperator_AbsoluteValueFloat64

Return the absolute value of the operand

4.6.2.53 dmaQueryOperator_MultiplyFloat64

Return the product of the operands

4.6.2.54 dmaQueryOperator_DivideFloat64

Divide the first operand by the second and return the quotient

4.6.2.55 dmaQueryOperator_Integer32ToFloat32

Convert the integer32 operand's value to a float64 and return the result

4.6.2.56 dmaQueryOperator_Float64ToInteger32Round

Convert the float64 operand's value to an integer32 by rounding it and return the result

4.6.2.57 dmaQueryOperator_Float64ToInteger32Truncate

Convert the float64 operand's value to an integer32 by truncating it and return the result

4.6.2.58 dmaQueryOperator_Exists

The subquery operand must be of type QueryRoot. It must have an empty SelectionsList, and therefore be of datatype Class. If the subquery finds any result rows, True is returned, else False

4.6.2.59 dmaQueryOperator_InBinary

The operand must be a query property node, a query constant node, or a subquery node. If the operand is of type QueryRoot, the subquery must have exactly one property in its SelectionsList, and it must be of type Binary. Hence, the datatype of the subquery will be Binary. The value of the first operand is compared against the list of Binary values returned by the subquery. True is returned if it is amongst them, else False

4.6.2.60 dmaQueryOperator_InBoolean

The operand must be a query property node, a query constant node, or a subquery node. If the operand is of type QueryRoot, the subquery must have exactly one property in its SelectionsList, and it must be of type Boolean. Hence, the datatype of the subquery will be Boolean. The value of the first operand is compared against the list of Boolean values returned by the subquery. True is returned if it is amongst them, else False

4.6.2.61 dmaQueryOperator_InString

The operand must be a query property node, a query constant node, or a subquery node. If the operand is of type QueryRoot, the subquery must have exactly one property in its SelectionsList, and it must be of type String. Hence, the datatype of the subquery will be String. The value of the first operand is compared against the list of String values returned by the subquery. True is returned if it is amongst them, else False

4.6.2.62 dmaQueryOperator_InInteger32

The operand must be a query property node, a query constant node, or a subquery node. If the operand is of type QueryRoot, the subquery must have exactly one property in its SelectionsList, and it must be of type Int32. Hence, the datatype of the subquery will be Int32. The value of the first operand is compared against the list of Int32 values returned by the subquery. True is returned if it is amongst them, else False

4.6.2.63 dmaQueryOperator_InFloat64

The operand must be a query property node, a query constant node, or a subquery node. If the operand is of type QueryRoot, the subquery must have exactly one property in its SelectionsList, and it must be of type Float64. Hence, the datatype of the subquery will be Float64. The value of the first operand is

compared against the list of Float64 values returned by the subquery. True is returned if it is amongst them, else False

4.6.2.64 dmaQueryOperator_InDateTime

The operand must be a query property node, a query constant node, or a subquery node. If the operand is of type QueryRoot, the subquery must have exactly one property in its SelectionsList, and it must be of type Datetime. Hence, the datatype of the subquery will be Datetime. The value of the first operand is compared against the list of Datetime values returned by the subquery. True is returned if it is amongst them, else False

4.6.2.65 dmaQueryOperator_InId

The operand must be a query property node, a query constant node, or a subquery node. If the operand is of type QueryRoot, the subquery must have exactly one property in its SelectionsList, and it must be of type ID. Hence, the datatype of the subquery will be ID. The value of the first operand is compared against the list of ID values returned by the subquery. True is returned if it is amongst them, else False

4.7 DMA Property DataTypes

The following are the property datatypes defined in DMA 1.0:

Name	Define Name	Description
Binary	DMA_DATATYPE_BINARY	Base data type DmaBinaryValue. Array of unsigned 8 bit bytes.
Boolean	DMA_DATATYPE_BOOLEAN	Base data type DmaBoolean. Can take on at least the values DMA_TRUE, DMA_FALSE, and DMA_UNKNOWN.
Date Time	DMA_DATATYPE_DATETIME	Base data type pDmaDateTime. A DMA string holding a date time in a format described in the DMA Internationalization and Localization section.
Float64	DMA_DATATYPE_FLOAT64	Base data type DmaFloat64. An IEEE standard floating point number.
ID	DMA_DATATYPE_ID	Base data type DmaId. A DCE UUID or Microsoft GUID.
Integer32	DMA_DATATYPE_INTEGER32	Base data type DmaInteger32. A signed integer that can take on at least the values -2,147,483,648 to +2,147,483,647.
Object	DMA_DATATYPE_OBJECT	Base data type Dmapv. A COM interface pointer to a DMA object.
String	DMA_DATATYPE_STRING	Base data type pDmaString. A DMA string. (See the DMA Internationalization and Localization section.)
Any Base Datatype	DMA_DATATYPE_ANY_BASE_DATATYPE	Not a base data type. Indicates that any base data type will be acceptable in the context in which it is used. It is used to specify the base data type of the operand to the operators such as dmaQueryOperator_IsNull and dmaQueryOperator_IsDefined.
Class	DMA_DATATYPE_CLASS	Not a base data type. This value is used as the result type for a join operator, and as the required data type in operand descriptions where a join expression, searchable class or subquery with empty select list is required.
Result Row	DMA_DATATYPE_RESULT_ROW	Not a base data type. This indicates that the value is a query result row object.

4.8 Conformance

The description of conformance is presented in two parts – the mandatory baseline that any implementation of the DMA API must provide, and a detailed definition of each optional capability.

4.8.1 Mandatory Baseline

4.8.1.1 General

Every DMA object describes itself by providing metadata, as specified in the object model overview.

All the DMA classes are described in the reference sections of this specification. For each DMA class, the specification defines what properties and interfaces must be supported for it, given that the class itself is supported.

In order to support self describing metadata, at least the following DMA classes (referred to as SDM classes below) must be supported:

- DMA
- Metadata
- Class Description
- Property Description
- Property Description Boolean
- Property Description Id
- Property Description Integer32
- Property Description Object
- Property Description String
- List
- List Of Boolean
- List Of Id
- List Of Integer32
- List Of Object
- List Of String

The DMA classes that must be supported by all implementations vary according to metadata space type.

The following are the metadata space types, and the minimum set of DMA classes that they must support:

4.8.1.1.1 System Metadata Spaces

- SDM classes
- System
- Enumeration
- Enumeration Of Object

4.8.1.1.2 Individual Document Space Metadata Spaces

- SDM classes
- DocSpace
- DocVersion

4.8.1.1.3 Individual Document Space Scope Metadata Spaces

- SDM classes
- Scope
- Query
- Query Constant
- Query Node
- Query Nonterminal Node
- Query Operand Description
- Query Operator
- Query Operator Description
- Query Order By Node
- Query Property
- Query Root
- Query Searchable Class

4.8.1.1.4 Merged Scope Metadata Spaces

- (Same as Individual Document Space Metadata Space)

4.8.1.1.5 Query Result Metadata Spaces

- SDM classes
- Enumeration
- Query Result Row
- Query Result Set

4.8.1.2 System Object

The system object is a DMA object implemented by software referred to as “middleware”. The main functions of the middleware are to provide an implementation of the system object, a means for registering and locating document spaces, and to provide merged scope objects generated from multiple document space scopes, so that cross repository queries can be performed.

Every conformant implementation of the system object must provide:

- implementation of all interfaces, methods, classes, and properties identified as required to be implemented by the DMA object in the reference sections of this specification.
- thread safety under unrestricted threading (in which multiple threads of a process may execute both user and OS code concurrently).

4.8.1.3 Document Repository

The main purpose of document repository implementations of the DMA API is to provide an implementation of the Document Space object, and objects generated from it, e.g., scope objects.

Every conformant document space implementation must provide:

- implementation of all interfaces, methods, classes, and properties identified as required to be implemented by document spaces in the reference sections of this specification.
- thread safety under free threading.
- document space specific properties and/or content for documents.
- at least read only access to all documents stored in the document space: Implementation of all interfaces, methods, classes, and properties necessary for at least one of the following options is required:

- Query access: Documents can be accessed using a search involving the values of properties of persistent objects.
- Containment access: Starting from the list of initial containers or a container in hand, containment related properties can be used to navigate to documents.
- OIID: The IdmaDocSpace:: ConnectObject method is supported, and can provide access to a document given its OIID.

4.8.2 Optional Capabilities

4.8.2.1 Document Space Object

In the subsections of this section, the identifiers for document space capabilities (e.g., DMAC_LEVEL, DMAC_AUTHENTICATE, etc.) appear in the DMA header files as macros that evaluate to unique small, nonnegative integers. The capability identifiers are intended to be used as a list element index of the Doc Space Capabilities property, which is a list of integer32 property native to the document space object (class Doc Space). When the following refers to a capability being “set” or “reset”, that means that the value of the corresponding list element is DMA_TRUE or DMA_FALSE, respectively.

4.8.2.2 General

4.8.2.2.1 DMAC_LEVEL

The value indicates the version level number of the capabilities property for the document repository. It is a positive integer equal to 1 for the 1.0 DMA spec. On every subsequent release of the DMA spec. in which additional document repository capabilities are introduced, this number is increased by one.

4.8.2.2.2 DMAC_AUTHENTICATE

If set, then the document space object supports the IdmaAuthentication interface.

If reset, the document space object does not support the IdmaAuthentication interface.

4.8.2.2.3 DMAC_WRITEABLE_DOCSPACE

If set, then (1) document space objects can be added, and deleted, and modified, and/or (2) document versions can be checked in.

If reset, the document space is read only.

4.8.2.2.4 DMAC_CONNECT_VIA_OIID

If set, IdmaDocSpace:: ConnectObject is supported on the document space object to connect to persistent objects given their OIID.

If reset, IdmaDocSpace:: ConnectObject is not supported.

4.8.2.3 Content

4.8.2.3.1 DMAC_CONTENT

This is set if and only if content is supported: At least some documents in the document space can have content. This implies that the Rendition class and at least one of the classes Content Transfer and Content Reference must be supported.

If reset, the document repository merely catalogs external material, e.g., books on a shelf, and no documents have content.

4.8.2.3.2 DMAC_CONTENT_MULT_RENDITIONS

If set, then content is supported, and documents can have multiple renditions.

If reset, documents can have at most one rendition.

4.8.2.3.3 DMAC_CONTENT_MULT_ELEMENTS

If set, then content is supported, and renditions can have multiple content elements.

If reset, renditions can not have more than one content element.

4.8.2.3.4 DMAC_CONTENT_MIME_RENDITIONS

If set, then content is supported, and the document space supports the use of rendition types from the MIME rendition type space. NOTE: The Rendition Type property is a string of the form <rendition_type_space>::<typename>. The only <rendition_type_space> defined in the DMA 1.0 spec. is "MIME" (Multipurpose Internet Mail Extension) as defined by the IANA. Examples are "MIME::application/postscript" and "MIME::text". END NOTE

If reset, then the document space does not support MIME reference type names.

4.8.2.3.5 DMAC_CONTENT_UPDATEABLE

If set, then document content is supported, and document content can be edited. Some or all of the following operations can be performed on existing persistent Doc Version objects: Renditions can be added; renditions can be deleted; renditions can be replaced; content elements can be added; content

elements can be deleted; content elements can be replaced. This implies IdmaConnection:: ExecuteChange is supported.

If reset, then document cannot be edited. (Note that checking in a new version of a document does not count as updating content in the sense of this capability. That is viewed as creating a new persistent Doc Version object for purposes of discussion of this capability.)

4.8.2.3.6 DMAC_CONTENT_CAPTURE_RESOURCE

If set, then content is supported, and IdmaContentTransfer:: SetCaptureResource supported. This method allows content to be captured from a named resource.

If reset, then IdmaContentTransfer:: SetCaptureResource is not supported.

4.8.2.3.7 DMAC_CONTENT_COPY_TO_RESOURCE

If set, then content is supported, and IdmaContentTransfer:: CopyToResource is supported. This method makes a copy of captured content data in a resource of a specified name.

If reset, then IdmaContentTransfer:: CopyToResource is not supported.

4.8.2.4 Containment

If containment is supported, the following optional classes must be supported:

- Containable
- Container
- Relationship

4.8.2.4.1 DMAC_CONTAIN_DIR_REQUIRED

If set, then, any directly containable object (except for root direct containers) must be contained in a container via direct containment. (There are no free-standing directly containable objects except root direct containers.)

If reset, then it is not required that directly containable objects be contained in a container via direct containment.

If containment is not implemented, the value is reset.

4.8.2.4.2 DMAC_CONTAIN_REF_REQUIRED

If set, then, any referentially containable object (except for root referential containers) must be contained in a container via referential containment. (There

are no freestanding referentially containable objects except root referential containers.)

If reset, then it is not required that referentially containable objects be contained in a container via referential containment.

If containment is not implemented, the value is reset.

If `DMAC_CONTAIN_REF_REQUIRED` and `DMAC_CONTAIN_DIR_REQUIRED` are both set, and if there is a class that can be contained both via direct containment and referential containment, then objects of that class must be directly contained, but need not be referentially contained.

4.8.2.4.3 DMAC_CONTAIN

If `DMAC_CONTAIN` is set, then containment is supported. The value of `DMAC_CONTAIN` is defined to be `DMAC_CONTAIN_REF` OR `DMAC_CONTAIN_DIR`.

4.8.2.4.4 DMAC_CONTAIN_REF

If set, then referential containment is supported. The value of `DMAC_CONTAIN_REF` is defined to be `DMAC_CONTAIN_REF_CONTAINER` OR `DMAC_CONTAIN_REF_CFG_HISTORY` OR `DMAC_CONTAIN_REF_VER_SERIES` OR `DMAC_CONTAIN_REF_VER_DESC` OR `DMAC_CONTAIN_REF_DOC_VER` OR `DMAC_CONTAIN_REF_OTHER`.

If `DMAC_CONTAIN_REF` is set, the following optional classes must be supported:

- Referential Containment Relationship

4.8.2.4.5 DMAC_CONTAIN_REF_CONTAINER

If set, then classes of type Container and subclasses of that type may be contained referentially.

If reset, classes of type Container and subclasses of that type may not be referentially contained.

4.8.2.4.6 DMAC_CONTAIN_REF_CFG_HISTORY

If set, then classes of type Configuration History and subclasses of that type may be contained referentially.

If reset, then classes of type Configuration History and subclasses of that type may not be referentially contained.

4.8.2.4.7 DMAC_CONTAIN_REF_VER_SERIES

If set, then classes of type Version Series and subclasses of that type may be contained referentially.

If reset, then classes of type Version Series and subclasses of that type may not be referentially contained.

4.8.2.4.8 DMAC_CONTAIN_REF_VERS_DESC

If set, then classes of type Version Description and subclasses of that type may be contained referentially.

If reset, then classes of type Version Description and subclasses of that type may not be referentially contained.

4.8.2.4.9 DMAC_CONTAIN_REF_DOC_VER

If set, then classes of type Doc Version and subclasses of that type may be contained referentially.

If reset, then classes of type Doc Version and subclasses of that type may not be referentially contained.

4.8.2.4.10 DMAC_CONTAIN_REF_OTHER

If set, then a class of type other than Doc Version, Configuration History, Version Series, and Version Description and subclasses of those classes may be contained referentially.

If reset, then no classes of type other than Doc Version, Configuration History, Version Series, and Version Description and subclasses of those classes may be contained referentially.

4.8.2.4.11 DMAC_CONTAIN_REF_MAX_DEPTH

This is the maximum number of recursive levels that containers can contain other containers referentially. If a container can not be contained referentially by another container, the value is zero. If there is no limit on the number of levels of referential containment of containers, the value is -1.

If referential containment is not supported, the value is zero.

4.8.2.4.12 DMAC_CONTAIN_REF_ORDER

The value of this capability is defined as DMAC_CONTAIN_REF_ORDER_HEAD OR DMAC_CONTAIN_REF_ORDER_TAIL .

4.8.2.4.13 DMAC_CONTAIN_REF_ORDER_HEAD

If set, the `IdmaRelationshipOrdering` interface is supported for Referential Containment Relationship objects, and the `SetOrdering` method accepts the value `DMA_POS_DEFAULT` for the value of the `ulHeadWhere` parameter.

If reset, the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects.

4.8.2.4.14 DMAC_CONTAIN_REF_ORDER_HEAD_BEFORE

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_BEFORE` for the `ulHeadWhere` parameter for Referential Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects, or the `DMA_POS_BEFORE` value for the `ulHeadWhere` parameter to `SetOrdering` is not supported for Referential Containment Relationship objects.

4.8.2.4.15 DMAC_CONTAIN_REF_ORDER_HEAD_AFTER

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_AFTER` for the `ulHeadWhere` parameter for Referential Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects, or the `DMA_POS_AFTER` value for the `ulHeadWhere` parameter to `SetOrdering` is not supported for Referential Containment Relationship objects.

4.8.2.4.16 DMAC_CONTAIN_REF_ORDER_HEAD_FIRST

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_FIRST` for the `ulHeadWhere` parameter for Referential Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects, or the `DMA_POS_FIRST` value for the `ulHeadWhere` parameter to `SetOrdering` is not supported for Referential Containment Relationship objects.

4.8.2.4.17 DMAC_CONTAIN_REF_ORDER_HEAD_LAST

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_LAST` for the `ulHeadWhere` parameter for Referential Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects, or the `DMA_POS_LAST` value

for the `ulHeadWhere` parameter to `SetOrdering` is not supported for Referential Containment Relationship objects.

4.8.2.4.18 DMAC_CONTAIN_REF_ORDER_HEAD_NO_CH

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_NO_CHANGE` for the `ulHeadWhere` parameter for Referential Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not implemented for Referential Containment Relationship objects, or the `DMA_POS_NO_CHANGE` value for the `ulHeadWhere` parameter to `SetOrdering` is not supported for Referential Containment Relationship objects.

4.8.2.4.19 DMAC_CONTAIN_REF_ORDER_TAIL

If set, the `IdmaRelationshipOrdering` interface is supported for Referential Containment Relationship objects, and the `SetOrdering` method accepts the value `DMA_POS_DEFAULT` for the value of the `ulTailWhere` parameter.

If reset, the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects.

4.8.2.4.20 DMAC_CONTAIN_REF_ORDER_TAIL_BEFORE

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_BEFORE` for the `ulTailWhere` parameter for Referential Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects, or the `DMA_POS_BEFORE` value for the `ulTailWhere` parameter to `SetOrdering` is not supported for Referential Containment Relationship objects.

4.8.2.4.21 DMAC_CONTAIN_REF_ORDER_TAIL_AFTER

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_AFTER` for the `ulTailWhere` parameter for Referential Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects, or the `DMA_POS_AFTER` value for the `ulTailWhere` parameter to `SetOrdering` is not supported for Referential Containment Relationship objects.

4.8.2.4.22 DMAC_CONTAIN_REF_ORDER_TAIL_FIRST

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_FIRST` for the `ulTailWhere` parameter for Referential Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects, or the `DMA_POS_FIRST` value for the `ulTailWhere` parameter to `SetOrdering` is not supported for Referential Containment Relationship objects.

4.8.2.4.23 DMAC_CONTAIN_REF_ORDER_TAIL_LAST

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_LAST` for the `ulTailWhere` parameter for Referential Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects, or the `DMA_POS_LAST` value for the `ulTailWhere` parameter to `SetOrdering` is not supported for Referential Containment Relationship objects.

4.8.2.4.24 DMAC_CONTAIN_REF_ORDER_TAIL_NO_CH

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_NO_CHANGE` for the `ulTailWhere` parameter for Referential Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Referential Containment Relationship objects, or the `SetOrdering DMA_POS_NO_CHANGE` value for the `ulTailWhere` parameter to `SetOrdering` is not supported for Referential Containment Relationship objects.

4.8.2.4.25 DMAC_CONTAIN_DIR

The value of `DMAC_CONTAIN_DIR` is defined to be `DMAC_CONTAIN_DIR_CONTAINER` OR `DMAC_CONTAIN_DIR_CFG_HISTORY` OR `DMAC_CONTAIN_DIR_VER_SERIES` OR `DMAC_CONTAIN_DIR_VER_DESC` OR `DMAC_CONTAIN_DIR_DOC_VER` OR `DMAC_CONTAIN_DIR_OTHER` .

If `DMAC_CONTAIN_DIR` is set, the following optional classes must be supported:

- Direct Containment Relationship

4.8.2.4.26 DMAC_CONTAIN_DIR_CONTAINER

If set, then classes of type `Container` and subclasses of that type may be contained directly.

If reset, then classes of type `Container` and subclasses of that type may not be directly contained.

4.8.2.4.27 DMAC_CONTAIN_DIR_CFG_HISTORY

If set, then classes of type Configuration History and subclasses of that type may be contained directly.

If reset, then classes of type Configuration History and subclasses of that type may not be directly contained.

4.8.2.4.28 DMAC_CONTAIN_DIR_VER_SERIES

If set, then classes of type Version Series and subclasses of that type may be contained directly.

If reset, then classes of type Version Series and subclasses of that type may not be directly contained.

4.8.2.4.29 DMAC_CONTAIN_DIR_VER_DESC

If set, then classes of type Version Description and subclasses of that type may be contained directly.

If reset, then classes of type Version Description and subclasses of that type may not be directly contained.

4.8.2.4.30 DMAC_CONTAIN_DIR_DOC_VER

If set, then classes of type Doc Version and subclasses of that type may be contained directly.

If reset, then classes of type Doc Version and subclasses of that type may not be directly contained.

4.8.2.4.31 DMAC_CONTAIN_DIR_OTHER

If set, then a class of type other than Doc Version, Configuration History, Version Series, and Version Description and subclasses of those classes may be contained directly.

If reset, then no classes of type other than Doc Version, Configuration History, Version Series, and Version Description and subclasses of those classes may be contained directly.

4.8.2.4.32 DMAC_CONTAIN_DIR_MAX_DEPTH

This is the maximum number of recursive levels that containers can contain other containers via direct containment. If a container can not be contained by another container via direct containment, the value is zero. If there is no limit on the number of levels, the value is -1.

If direct containment is not supported, the value is zero.

4.8.2.4.33 DMAC_CONTAIN_DIR_ORDER

This capability is defined as DMAC_CONTAIN_DIR_ORDER_HEAD OR DMAC_CONTAIN_DIR_ORDER_TAIL .

4.8.2.4.34 DMAC_CONTAIN_DIR_ORDER_HEAD

If set, the IdmaRelationshipOrdering interface is supported for Direct Containment Relationship objects, and the SetOrdering method accepts the value DMA_POS_DEFAULT for the value of the ulHeadWhere parameter.

If reset, the IdmaRelationshipOrdering interface is not supported for Direct Containment Relationship objects.

4.8.2.4.35 DMAC_CONTAIN_DIR_ORDER_HEAD_BEFORE

If set, the IdmaRelationshipOrdering:: SetOrdering method allows the value DMA_POS_BEFORE for the ulHeadWhere parameter for Direct Containment Relationship objects.

If reset, either the IdmaRelationshipOrdering interface is not supported for Direct Containment Relationship objects, or the DMA_POS_BEFORE value for the ulHeadWhere parameter to SetOrdering is not supported for Direct Containment Relationship objects.

4.8.2.4.36 DMAC_CONTAIN_DIR_ORDER_HEAD_AFTER

If set, the IdmaRelationshipOrdering:: SetOrdering method allows the value DMA_POS_AFTER for the ulHeadWhere parameter for Direct Containment Relationship objects.

If reset, either the IdmaRelationshipOrdering interface is not supported for Direct Containment Relationship objects, or the DMA_POS_AFTER value for the ulHeadWhere parameter to SetOrdering is not supported for Direct Containment Relationship objects.

4.8.2.4.37 DMAC_CONTAIN_DIR_ORDER_HEAD_FIRST

If set, the IdmaRelationshipOrdering:: SetOrdering method allows the value DMA_POS_FIRST for the ulHeadWhere parameter for Direct Containment Relationship objects.

If reset, either the IdmaRelationshipOrdering interface is not supported for Direct Containment Relationship objects, or the DMA_POS_FIRST value for the ulHeadWhere parameter to SetOrdering is not supported for Direct Containment Relationship objects.

4.8.2.4.38 DMAC_CONTAIN_DIR_ORDER_HEAD_LAST

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_LAST` for the `ulHeadWhere` parameter for Direct Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Direct Containment Relationship objects, or the `DMA_POS_LAST` value for the `ulHeadWhere` parameter to `SetOrdering` is not supported for Direct Containment Relationship objects.

4.8.2.4.39 DMAC_CONTAIN_DIR_ORDER_HEAD_NO_CH

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_NO_CHANGE` for the `ulHeadWhere` parameter for Direct Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not implemented for Direct Containment Relationship objects, or the `DMA_POS_NO_CHANGE` value for the `ulHeadWhere` parameter to `SetOrdering` is not supported for Direct Containment Relationship objects.

4.8.2.4.40 DMAC_CONTAIN_DIR_ORDER_TAIL

If set, the `IdmaRelationshipOrdering` interface is supported for Direct Containment Relationship objects, and the `SetOrdering` method accepts the value `DMA_POS_DEFAULT` for the value of the `ulTailWhere` parameter.

If reset, the `IdmaRelationshipOrdering` interface is not supported for Direct Containment Relationship objects.

4.8.2.4.41 DMAC_CONTAIN_DIR_ORDER_TAIL_BEFORE

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_BEFORE` for the `ulTailWhere` parameter for Direct Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Direct Containment Relationship objects, or the `DMA_POS_BEFORE` value for the `ulTailWhere` parameter to `SetOrdering` is not supported for Direct Containment Relationship objects.

4.8.2.4.42 DMAC_CONTAIN_DIR_ORDER_TAIL_AFTER

If set, the `IdmaRelationshipOrdering::SetOrdering` method allows the value `DMA_POS_AFTER` for the `ulTailWhere` parameter for Direct Containment Relationship objects.

If reset, either the `IdmaRelationshipOrdering` interface is not supported for Direct Containment Relationship objects, or the `DMA_POS_AFTER` value for the

ulTailWhere parameter to SetOrdering is not supported for Direct Containment Relationship objects.

4.8.2.4.43 DMAC_CONTAIN_DIR_ORDER_TAIL_FIRST

If set, the IdmaRelationshipOrdering:: SetOrdering method allows the value DMA_POS_FIRST for the ulTailWhere parameter for Direct Containment Relationship objects.

If reset, either the IdmaRelationshipOrdering interface is not supported for Direct Containment Relationship objects, or the DMA_POS_FIRST value for the ulTailWhere parameter to SetOrdering is not supported for Direct Containment Relationship objects.

4.8.2.4.44 DMAC_CONTAIN_DIR_ORDER_TAIL_LAST

If set, the IdmaRelationshipOrdering:: SetOrdering method allows the value DMA_POS_LAST for the ulTailWhere parameter for Direct Containment Relationship objects.

If reset, either the IdmaRelationshipOrdering interface is not supported for Direct Containment Relationship objects, or the DMA_POS_LAST value for the ulTailWhere parameter to SetOrdering is not supported for Direct Containment Relationship objects.

4.8.2.4.45 DMAC_CONTAIN_DIR_ORDER_TAIL_NO_CH

If set, the IdmaRelationshipOrdering:: SetOrdering method allows the value DMA_POS_NO_CHANGE for the ulTailWhere parameter for Direct Containment Relationship objects.

If reset, either the IdmaRelationshipOrdering interface is not supported for Direct Containment Relationship objects, or the SetOrdering DMA_POS_NO_CHANGE value for the ulTailWhere parameter to SetOrdering is not supported for Direct Containment Relationship objects.

4.8.2.5 Versioning

If versioning is supported, the following optional classes must be supported:

- Versionable
- Configuration History
- Version Series (Note that IdmaVersionSeries is a required interface on this class.)
- Version Description

If versioning is supported, the following optional interfaces must be supported by the Doc Version class:

- IdmaVersionable

4.8.2.5.1 DMAC_VERSION

If set, new document versions can be created via IdmaVersionable:: Set-Checkin on the Doc Version object, followed by IdmaConnection:: ExecuteChange on the Doc Version object..

If reset, the checking in of new document versions is not supported.

4.8.2.5.2 DMAC_VERSION_REQUIRED

If set, then the versioning related properties are required to have values. Thus, for example, all documents have to be versioned.

If reset, then the versioning related properties are not required to have values. Thus, for example, there can be documents not in a version series.

4.8.2.5.3 DMAC_VERSION_OTHER_ENTITIES

If set, then there are classes that are not subclasses of Versionable that have versioning properties, and so may be versioned.

If reset, then only classes that are subclasses of Versionable can have versioning properties.

4.8.2.5.4 DMAC_VERSION_CHECKIN_EXISTING

If set, then the document space allows checkin of an object that is already persistent.

If reset, then the document space does not allow checkin of an object that is already persistent.

4.8.2.5.5 DMAC_VERSION_THREADED

If set, then a Doc Version can have more than one Version Description as a parent.

If reset, then a Doc Version can have at most one Version Description as a parent.

4.8.2.5.6 DMAC_VERSION_CHECKOUT

If set, then the document space supports IdmaVersionSeries:: SetCheckOut-Next.

If reset, then the document space does not support SetCheckOutNext. (SetReserveNext is in the versioning baseline. SetCheckOutNext isn't.)

4.8.2.5.7 DMAC_VERSION_CHANGE_CURRENT

If set, then the document space supports making a version other than the last be the current version. The current version is what gets checked out and leads to the next version when it's checked in. Changing the current version would be done by changing the Current Version Description property of the Version Series object.

If reset, the current version cannot be modified. This implies that the current version is always the last version. This also implies that the only way to insert a new version is checkout or reserve, followed by checkin.

4.8.2.5.8 DMAC_VERSION_DELETE_VERSION_DESCRIPTION

If set, then the document space supports deleting some, but not necessarily all, Version Descriptions in one operation.

If reset, versions can not be deleted selectively.

4.8.2.6 Query

4.8.2.6.1 DMAC_QUERY

If set, then query is supported: IdmaDocSpace:: GetScope is supported, and at least the following query classes are supported: Scope, Query, Query Node, Query Root, Query Property, Query Searchable Class, Query Result Set, Query Result Row. There is at least one searchable class in the Searchable Classes property of scopes generated by the document space, and at least one selectable property in every searchable class in scopes generated by the document space.

If reset, there are no useful queries that are supported. The IdmaDocSpace:: GetScope method is not supported, and none of the query classes are supported. The query classes are: Query, Query Constant, Query Constant Bina-ries, QueryConstant Binary, Query Constant Boolean, Query Constant Booleans, Query Constant Date Time, Query Constant Date Times, Query Constant Float64, Query Constant Float64s, Query Constant Id, Query Con-stant Ids, Query Constant Integer32, Query Constant Integer32s, Query Con-stant String, Query Constant Strings, Query Join Op, Query Node, Query Nonterminal Node, Query Operand Description, Query Operator, Query Oper-ator Description, Query Order By Node, Query Property, Query Result Row, Query Result Set, Query Root, Query Searchable Class, Scope.

4.8.2.6.2 DMAC_QUERY_EXPRESSION

If set, then the following is true: DMAC_QUERY is set. The Query Expression property of the Query Root object may be non null. The classes Query Operator Description, Query Operand Description, Query Nonterminal Node, Query Constant, and Query Operator are supported. At least one query operator that is not a join operator is supported. At least one of the following query classes is supported: Query Constant Binaries, Query Constant Binary, Query Constant Boolean, Query Constant Booleans, Query Constant Date Time, Query Constant Date Times, Query Constant Float64, Query Constant Float64s, Query Constant Id, Query Constant Ids, Query Constant Integer32, Query Constant Integer32s, Query Constant String, Query Constant Strings.

If reset, then the Query Expression property of the Query Root object must be null.

4.8.2.6.3 DMAC_QUERY_JOINS

If set, then DMAC_QUERY is set, and at least one query join operator is supported.

If reset, then no query join operators are supported.

4.8.2.6.4 DMAC_QUERY_ORDERINGS

If set, then DMA_QUERY is set, and the Orderings property of the Query Root object may be non null.

If reset, then the Orderings property of the Query Root object must be null.

4.8.2.6.5 DMAC_3V_ELIMINATION

If set, Scope objects created by the current document space object always perform three valued elimination on queries when requested to do so.

If reset, Scope objects created by the current document space object never perform three valued elimination on queries.

4.8.2.6.6 DMAC_LOCKING

If this capability is set, locking is supported. The value of DMAC_LOCKING is defined as DMAC_LOCKING_WRITER OR DMAC_LOCKING_READER OR DMAC_LOCKING_EXISTENCE.

Required Interfaces and methods:

- IdmaConnection
 - ApplyLock
 - RemoveLock

- IdmaDocSpace
 - ConnectAndLockObject

4.8.2.6.7 DMAC_LOCKING_WRITER

If set, the value DMA_LOCK_WRITE for the ILockType parameter to ApplyLock is supported.

If reset, the value DMA_LOCK_WRITE for the ILockType parameter to ApplyLock is not supported.

4.8.2.6.8 DMAC_LOCKING_READER

If set, the value DMA_LOCK_READ for the ILockType parameter to ApplyLock is supported.

If reset, the value DMA_LOCK_READ for the ILockType parameter to ApplyLock is not supported.

4.8.2.6.9 DMAC_LOCKING_EXISTENCE

If set, the value DMA_LOCK_EXIST for the ILockType parameter to ApplyLock is supported.

If reset, the value DMA_LOCK_EXIST for the ILockType parameter to ApplyLock is not supported.

5 DMA Appendix

The code examples and glossary included in this appendix are for illustrative purposes only and are not part of the formal DMA specification.

- System & Document Space Connection Example
- Content Examples
- Pretty Print Example
- Query Examples
- Containment Examples
- Versioning Example
- DMA Reference Header Files and Samples (You can retrieve the files from <http://www.aiim.org/dma/dma10/dmasrc.zip>.)
- DMA Glossary

5.1 System & Document Space Connection Example

```

/*
 * Copyright 1995,1996,1997 by the Association for Information and Image Man-
agement Int'l
 *
 *                               1100 Wayne Avenue
 *                               Silver Spring, MD 20910-5603
 *                               Tel: 301/587-8202
 *                               Fax: 301/587-2711
 *
 * All Rights Reserved.
 * DMA (Document Management Alliance) working group.
 *
 * Code example: System & Document Space Connection Example
 *
 */

#include <dmatypes.h>
#include <dmacom.h>
#include <dmarc.h>
#include <dmaenums.h>
#include <dmaiface.h>
#include <dmaenums.h>
#include <dmaids.h>
#include <dmaidvar.h>
#include <dmacfunc.h>

#ifdef NULL
#define NULL    0
#endif

#define MAXSYSTEMS 100           // arbitrary
#define MAXDOCSPACE_PERSYSTEM 100 // Maximums

// external functions
DmaRC FreeString(pDmaString pDmaStr);
pDmaString AsciiDmaStringFromCString(char *string);

// forward declarations
DmaRC Authenticate ( IUnknown *punk, char *LoginStr );

/*
 * This example is comprised of 2 loops; an outer system loop,
 * and an inner document space loop
 *
 * An enumerated list of systems that are connected to the system manager
 * is created.
 *
 * For each system (the outer loop) the Display name is obtained,
 * and an authentication is performed.
 */

```

```

* Inside this outer loop an enumeration of all the document spaces is
* obtained.
*
* For each document space in that system(the inner loop) the Display name
* is obtained, and an authentication is performed
*/

DmaRC IntegrationExample()
{
    DmaRC          rc;
    pDmaString      pSystemDisplayName, pDocSpaceDisplayName;
    IdmaSystemManager *pISysMgr;
    IdmaEnumOfObject *pIEnumSystems, *pIEnumDocSpaces;
    DmaIndex32      NumSystem, NumDocSpace;
    IdmaProperties   *pIProps;
    DmaIndex32      iSystem, iDocSpace;
    char *SystemLoginString = "SCHEME=DMA-BASIC;UID=SYSTEM;PWD=MANAGER";
    char *DocSpaceLoginString = "SCHEME=DMA-BASIC;UID=SCOTT;PWD=TIGER";

    /*
     * Get a IdmaSystem Manager Interface
     */
    rc = dmaConnectSystemManager( NULL,          // Environment info
                                NULL,          // System config location
                                NULL,          // IMalloc Interface
                                DMA_CS_UNICODE, // Char Set Coding ID
                                NULL,          // Name Of Display
                                IID_IdmaSystemManager,
                                // Desired Interface
                                (pDmapv) &pISysMgr);

    if (rc != DMARC_OK)
    {
        return(rc);
    }

    // All future error handling omitted for simplicity

    /*
     * Enumerate the Systems Connected To
     */
    rc = pISysMgr->EnumerateSystems(IID_IdmaEnumOfObject,
                                   (pDmapv) &pIEnumSystems);

    IUnknown * rgpunkSystems[MAXSYSTEMS];

    rc = pIEnumSystems->GetNextObject ( MAXSYSTEMS, rgpunkSystems, &NumSystem
);
    pIEnumSystems->Release(); // no longer needed

    /*

```

```

    * Loop through systems and attempt to find the name of a system
    */

for (iSystem = 0; iSystem < NumSystem; iSystem++)
{
    /*
     * Get System's Property Interface
     */
    rc = rgpunkSystems[iSystem]->QueryInterface( IID_IdmaProperties,
                                                (pDmapv)&pIProps );

    /*
     * Get Display Name
     */
    rc = pIProps->GetPropValStringById ( &dmaProp_DisplayName,
                                        &pSystemDisplayName );

    pIProps->Release();          // No Longer Needed

    // Authenticate if necessary
    rc = Authenticate ( rgpunkSystems[iSystem], SystemLoginString );

    FreeString(pSystemDisplayName);

    /*
     * Enumerate the DocSpaces Available for this system
     */
    IdmaSystem *pISystem;
    rc = rgpunkSystems[iSystem]->QueryInterface ( IID_IdmaSystem,
                                                (pDmapv)&pISystem );

    rc = pISystem->EnumerateDocSpaces( IID_IdmaEnumOfObject,
                                      (pDmapv) &pIEnumDocSpaces);

    pISystem->Release();
    rgpunkSystems[iSystem]->Release();

    IUnknown *rgpunkDocSpaces[MAXDOCSPACE_PERSYSTEM];

    rc = pIEnumDocSpaces->GetNextObject ( MAXDOCSPACE_PERSYSTEM,
                                        rgpunkDocSpaces,
                                        &NumDocSpace );
    pIEnumDocSpaces->Release(); // no longer needed

    /*
     * Loop through Doc Spaces in this System
     */

    for (iDocSpace = 0; iDocSpace < NumDocSpace; iDocSpace++)
    {
        rc = rgpunkDocSpaces[iDocSpace]->QueryInterface (
                                                    IID_IdmaProperties,
                                                    (pDmapv) &pIProps );
    }
}

```

```

        rc = pIProps->GetPropValStringById ( &dmaProp_DisplayName,
                                            &pDocSpaceDisplayName );
        pIProps->Release();                // No Longer Needed

        // Authenticate if necessary
        rc = Authenticate ( rgpunkDocSpaces[iDocSpace],
                            DocSpaceLoginString );

        rgpunkDocSpaces[iDocSpace]->Release();    // No Longer Needed

        FreeString(pDocSpaceDisplayName);

    } // while no more Doc Spaces enumerated

} // while no more Systems enumerated

return(DMARC_OK);
}

DmaRC Authenticate (
    IUnknown *punk,
    char *LoginStr )
{
    IdmaAuthentication *pIAuth;

    DmaRC rc = punk->QueryInterface ( IID_IdmaAuthentication,
                                     (pDmapv)&pIAuth );

    if ( rc == DMARC_OK )
    {
        // The authentication interface is supported. Therefore we must
        // try to authenticate.
        pDmaString StringIn;
        pDmaString StringOut;

        StringIn = AsciiDmaStringFromCString(LoginStr);

        do
        {
            // Iteratively attempt to Authenticate
            rc = pIAuth->AuthenticateUser ( StringIn, &StringOut );

            FreeString(StringIn); // no longer needed

            if ( rc == DMARC_ALREADY_AUTHENTICATED ||
                rc == DMARC_OK )
            {
                FreeString(StringOut);
                return(DMARC_OK);
            }
        }
    }
}

```



```
        if (rc == DMARC_NEED_MORE_DATA)
        {
            /*
             * Note:At this point the Application would display
             * the information in the StringOut and get the
             * additional parameters needed in LoginStr via
             * user interface.
             */

            StringIn = AsciiDmaStringFromCString(LoginStr);
            FreeString(StringOut);
        }

        } while (rc == DMARC_NEED_MORE_DATA);

        pIAuth->Release();
    }
    else if ( rc == DMARC_BAD_INTERFACE )
    {
        // The authentication interface is not supported, meaning that
        // it is not necessary to authenticate.
        rc = DMARC_OK;
    }

    return(rc);
}
```

5.2 Content Examples

- CTEDIT.CPP demonstrates modifying the content of a document within a DMA repository.
- CTNEWDOC.CPP demonstrates creating a new document within a DMA repository.
- CTGETDOC.CPP demonstrates accessing the content of a document stored within a DMA repository

5.2.1 CTEDIT.CPP Example File

```
/*
 * Copyright 1995,1996,1997 by the Association for Information and Image Man-
agement Int'l
 *
 *                               1100 Wayne Avenue
 *                               Silver Spring, MD 20910-5603
 *                               Tel: 301/587-8202
 *                               Fax: 301/587-2711
 *
 * All Rights Reserved.
 * DMA (Document Management Alliance) working group.
 *
 * Code example: editing a document in DMA 1.0.
 */

#ifndef NULL
#define NULL    0
#endif

// ANSI C header file
#include <stddef.h>

// DMA header files
#include <dmatypes.h>
#include <dmastr.h>
#include <dmaenums.h>
#include <dmarc.h>
#include <dmacom.h>
#include <dmaiface.h>
#include <dmaids.h>

#define DMA_INIT_ID
#include <dmaidvar.h>
```

```
#define sampleProp_AuthorVal {0x0L, 0xBADL, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}

DMA_DECL_ID(sampleProp_Author)

// Code example: editing a document in DMA 1.0.

// A completely artificial example that changes the "author" property
// on an existing document and adds a new PDF rendition or replaces
// the one that's there.

// Assumes that connections to system and doc space have already been
// made, that the doc id of the target document has been found
// (by search or navigation), and that documents are not versioned.

// Assumes the doc ver has a DmaString-valued property "author".

// forward refs to helper routines
LPIdmaEditProperties MakeRendition(DmaWChar *,DmaWChar *);
DmaRC ReportError(DmaRC, char *msg);

// Utilities assumed to exist
extern DmaBoolean EqualDmaString(pDmaString, pDmaString);

// Assume these interface pointers have been initialized
LPMALLOC iMalloc;
LPIdmaSystem iSystem; /* interface on system object */
LPIdmaDocSpace iDocSpace; /* interface on the connected doc space */

Abort(DmaRC rc, char *msg)
{
    return(rc);
}

// Procedure to do the edits. Last argument is a URL giving path name
// to the PDF file that is to become the new rendition

EditStuff(pDmaString documentID, DmaWChar* newAuthor, DmaWChar* pdfFile)
{
    DmaRC rc;
    IdmaEditProperties* iDocProps = NULL;
    IdmaEditProperties* iRenditionObj = NULL;
    IdmaListOfString* iRenditionsPresent = NULL;
    IdmaEditListOfObject* iRenditions = NULL;
```

```
IdmaList*          iRendList = NULL;
IdmaConnection*    iDocConnection = NULL;
DmaIndex32         nRenditions;
pDmaString         authorString = NULL;
pDmaString         PDFTYPE = NULL;
pDmaString         renditionType = NULL;

// Connect to the document version object. This makes a snapshot
// of the persistent doc ver in the client's process space.

rc = iDocSpace->ConnectObject (documentID, IID_IdmaEditProperties,
                              (pDmapv) &iDocProps);

if (rc != DMARC_OK)
    Abort (rc, "Failed to connect to doc ver");

// NOTE: for brevity, further error checking is omitted. All DMA calls
// should be followed by code similar to the above

// Change the author property
if (newAuthor != NULL)
{
    DMA_CREATE_STRING (iMalloc->Alloc, 21, newAuthor, &authorString);
    rc = iDocProps->PutPropValStringById(&sampleProp_Author, authorString);
}

// Make our PDF rendition file into a DMA rendition object.
DMA_CREATE_STRING (iMalloc->Alloc, 21, L"MIME::application/PDF", &PDFTYPE);
iRenditionObj = MakeRendition(pdfFile, PDFTYPE);
if (iRenditionObj == NULL)
    Abort(DMARC_NO_MEMORY , "MakeRendition Failed");

// Get the RenditionsPresent property and check to see if a PDF
// rendition is present. Since IdmaListOfString inherits from
// IdmaList, it also has a GetCount method.
rc = iDocProps->GetPropValObjectById(&dmaProp_RenditionsPresent,
                                     IID_IdmaListOfString, (pDmapv) iRenditionsPresent);
rc = iRenditionsPresent->GetElementCount(&nRenditions);

int noPDF = 1;
rc = iDocProps->GetPropValObjectById(&dmaProp_Renditions,
                                     IID_IdmaEditListOfObject, (pDmapv) iRenditions);
for (int i = 0; i < nRenditions; i++)
{
    rc = iRenditionsPresent->GetString(i, &renditionType);
```

```
        if (EqualDmaString(renditionType, PDFTYPE))
        {
            rc =iRenditions->ReplaceObject(i, &iRenditionObj);
            noPDF = 0;
        }
        DMA_FREE_STRING(iMalloc->Free, renditionType);
    }
    if (noPDF)
    {
        rc =iRenditions->InsertObject(nRenditions, &iRenditionObj);
    }
    // Make the changes persistent

    rc = iDocProps->QueryInterface(IID_IdmaConnection, (pDmapv) &iDocConnection);
    rc = iDocConnection->ExecuteChange(NULL,DMA_MODIFY_UNPROTECTED,DMA_FALSE);

    // cleanup
    if (iDocProps != NULL) iDocProps->Release();
    if (iRenditionObj != NULL) iRenditionObj->Release();
    if (iRenditions != NULL) iRenditions->Release();
    if (iRenditionsPresent != NULL) iRenditionsPresent->Release();
    if (iDocConnection != NULL) iDocConnection->Release();

    DMA_FREE_STRING(iMalloc->Free, authorString);
    DMA_FREE_STRING(iMalloc->Free, PDFTYPE);
    return(DMARC_OK);
}

// Create a DMA rendition object of the indicated type. The rendition
// will have one content transfer element, whose content is named by the URL
// indicated by contentURL.

// Returns a pointer to the IdmaEditProperties interface on the new
// rendition, or NULL if the process failed for any reason.

// Ownership of the returned object and its content element and DMAStrings
// passes to the caller.

IdmaEditProperties* MakeRendition(DmaWChar* contentURL, DmaWChar* rendition-
Type)
{
    IdmaEditProperties*      iContentEl  = NULL;
    IdmaEditProperties*      iRenditionProps = NULL;
    IdmaContentTransfer*     iContentTr  = NULL;
```

```
IdmaEditListOfObject*   iContentEls = NULL;
IdmaObjectFactory*      iObjFactory = NULL;
pDmaString              contentURLString = NULL;
pDmaString              renditionType = NULL;
pDmaString              componentType = NULL;

// check arguments and convert to DMAStrings
if ((contentURL == NULL) || (wcslen(contentURL) == 0) ||
    (renditionType == NULL) || (wcslen(renditionType) == 0))
    return (NULL);
else
{
    DMA_CREATE_STRING (iMalloc->Alloc, wcslen(renditionType), renditionType,
&renditionType);
    DMA_CREATE_STRING (iMalloc->Alloc, wcslen(contentURL), contentURL, &contentURLString);
}

// Make content transfer element
DmaRC rc = iDocSpace->QueryInterface(IID_IdmaObjectFactory,
                                     (pDmapv) &iObjFactory);

if (rc != DMARC_OK)
    return (NULL);

// error checks omitted on following calls for brevity
rc = iObjFactory->CreateObject(&dmaClass_ContentTransfer,
                             IID_IdmaEditProperties, (pDmapv) &iContentEl);

// Set properties of the content element.
DMA_CREATE_STRING (iMalloc->Alloc, 6, L"single", &componentType);
rc = iContentEl->PutPropValStringById(&dmaProp_ComponentType, componentType);
rc = iContentEl->QueryInterface(IID_IdmaContentTransfer, (pDmapv) &iContentTr);
rc = iContentTr->SetCaptureResource(contentURLString);

// Make a rendition and add the content element to it
rc = iObjFactory->CreateObject(&dmaClass_Rendition, IID_IdmaEditProperties,
                             (pDmapv)& iRenditionProps);
rc = iRenditionProps->PutPropValStringById(&dmaProp_RenditionType, renditionType);
rc = iRenditionProps->GetPropValObjectById(&dmaProp_ContentElements,
                                           IID_IdmaEditListOfObject, (pDmapv)&iContentEls);
rc = iContentEls-> InsertObject(0, iContentEl);

// cleanup
if (iContentEl != NULL) iContentEl->Release();
if (iContentTr != NULL) iContentTr->Release();
```

```
if (iContentEls != NULL) iContentEls->Release();
if (iRenditionProps != NULL) iRenditionProps->Release();
if (iObjFactory != NULL) iObjFactory->Release();

DMA_FREE_STRING(iMalloc->Free, rendType);
DMA_FREE_STRING(iMalloc->Free, componentType);
DMA_FREE_STRING(iMalloc->Free, contentURLString);
return (iRenditionProps);
}
```

5.2.2 CTNEWDOC.CPP Example File

```
/*
 * Copyright 1995,1996,1997 The Association for Information and Image Manage-
ment Int'l
 *
 * 1100 Wayne Avenue
 * Silver Spring, MD 20910-5603
 * Tel: 301/587-8202
 * Fax: 301/587-2711
 *
 * All Rights Reserved.
 * DMA (Document Management Alliance) working group.
 *
 * Code example: Creation of a new simple document in DMA 1.0.
 */

#ifndef NULL
#define NULL 0
#endif

// ANSI C header file
#include <stddef.h>

// DMA header files
#include <dmatypes.h>
#include <dmastr.h>
#include <dmaenums.h>
#include <dmarc.h>
#include <dmacom.h>
#include <dmaiface.h>
#include <dmaids.h>
#include <dmaidvar.h>

// Assumes:
```

```
// connection to doc space has already been made
// documents are single rendition, single content element
// documents are not versioned.

extern LPIdmaDocSpace iDocSpace; /* interface on the connected doc space */

// Utilities assumed to exist
extern pDmaString MakeDmaString(DmaWChar*);
extern void FreeDmaString(pDmaString);

Abort(DmaRC rc, char *msg);

DmaWChar *docURL = L"file://\\project\\foo.txt"; /* URL of content file */

// While this example builds the document structure bottom up (from
// content transfer element up to doc ver), the reverse could also be done.

void StoreDocInDocSpace(DmaWChar *docURL)
{
    IdmaObjectFactory*      iObjectFactory = NULL;
    IdmaContentTransfer*    iContent = NULL;
    IdmaListOfObject*       iContentEls = NULL;
    IdmaEditListOfObject*   iEditContentList = NULL;
    IdmaEditListOfObject*   iEditRenditionList = NULL;
    IdmaConnection*         iDocConnection = NULL;
    IdmaEditProperties*      iDocVer = NULL;
    IdmaEditProperties*      iContentProps = NULL;
    IdmaEditProperties*      iRenditionProps = NULL;

    pDmaString docURLString = MakeDmaString(docURL);
    pDmaString single = MakeDmaString(L"single");
    pDmaString plainText = MakeDmaString(L"MIME::text/plain");

    // Create a content transfer element, and set its Capture Resource.
    // NOTE: the object factory in the doc space is used to create
    // the objects that are used in storing a new document -- since
    // those objects may be doc space dependent

    DmaRC rc = iDocSpace->QueryInterface(IID_IdmaObjectFactory, (pDmapv) &iObjectFactory);
    if (rc != DMARC_OK)
        Abort(rc, "Failed to get object factory interface");

    // NOTE: for brevity, further error checking is omitted.
```



```
// All DMA calls should be followed by code similar to the above

rc = iObjectFactory->CreateObject(&dmaClass_ContentTransfer,
                                   IID_IdmaContentTransfer, (pDmapv) &iContent);
rc = iContent->SetCaptureResource(docURLString);

// set required properties on the content transfer object
rc = iContent->QueryInterface(IID_IdmaEditProperties, (pDmapv) &iContentProps);
rc = iContentProps->PutPropValStringById(&dmaProp_ComponentType, single);

// Create a rendition, and link in content transfer object
rc = iObjectFactory->CreateObject(&dmaClass_Rendition,
                                   IID_IdmaProperties, (pDmapv) &iRenditionProps);
rc = iRenditionProps->GetPropValObjectById(&dmaProp_ContentElements,
                                           IID_IdmaEditListOfObject, (pDmapv) &iEditContentList);
rc = iEditContentList->InsertObject(0, (pDmapv) &iContent);

// Set required property - Rendition Type - on the rendition object
rc = iRenditionProps->PutPropValStringById(&dmaProp_RenditionType, plainText);

// Now make a doc version object and insert the rendition onto its rendition
list
rc = iObjectFactory->CreateObject(&dmaClass_DocVersion,
                                   IID_IdmaEditProperties, (pDmapv) &iDocVer);
// NOTE: since IdmaEditProperties inherits from IdmaProperties,
// no QueryInterface is needed
rc = iDocVer->GetPropValObjectById(&dmaProp_Renditions, IID_IdmaEditListOfOb-
                                   ject,
                                   (pDmapv) &iEditRenditionList);
rc = iEditRenditionList->InsertObject(0, (pDmapv) &iRenditionProps);

// NOTE: if we were using a sub-class of doc version with properties
// like title, author, ... we could set them here

// Make the new document persistent
rc = iDocVer->QueryInterface(IID_IdmaConnection, (pDmapv) &iDocConnection);
rc = iDocConnection->ExecuteChange(NULL, NULL, DMA_FALSE);

// cleanup
if (iObjectFactory != NULL) iObjectFactory->Release();
if (iContent != NULL) iContent->Release();
if (iContentEls != NULL) iContentEls->Release();
if (iEditContentList != NULL) iEditContentList->Release();
if (iEditRenditionList != NULL) iEditRenditionList->Release();
if (iDocConnection != NULL) iDocConnection->Release();
```

```
if (iDocVer != NULL) iDocVer->Release();
if (iContentProps != NULL) iContentProps->Release();
if (iRenditionProps != NULL) iRenditionProps->Release();

FreeDmaString(docURLString);
FreeDmaString(single);
FreeDmaString(plainText);
}
```

5.2.3 CTGETDOC.CPP Example File

```
/*
 * Copyright 1995,1996,1997 by the Association for Information and Image Man-
 * agement Int'l
 *
 *                               1100 Wayne Avenue
 *                               Silver Spring, MD 20910-5603
 *                               Tel: 301/587-8202
 *                               Fax: 301/587-2711
 *
 * All Rights Reserved.
 * DMA (Document Management Alliance) working group.
 *
 *
 * Code example: retrieval of document content in DMA 1.0.
 *
 */

#ifndef NULL
#define NULL    0
#endif

// ANSI C header file
#include <stddef.h>

// DMA header files

#include <dmatypes.h>
#include <dmastr.h>
#include <dmaenums.h>
#include <dmarc.h>
#include <dmacom.h>
#include <dmaiface.h>
#include <dmaids.h>
#include <dmaidvar.h>

// Code example: retrieval of document content in DMA 1.0.
```

```
// Assumes that connections to system and doc space have already been made
// and that the doc id of the target document has been found
// (by search or navigation)

extern LPIdmaDocSpace iDocSpace; // Ref to interface for the connected doc
space
extern LPMALLOC iMalloc; // An IMalloc interface obtained from IdmaSystem-
Manager::GetMalloc

// Utility that is assumed to exist
extern Abort(DmaRC rc, char *msg);

// DmaString utilities

pDmaString MakeDmaString(DmaWChar* content)
{
    pDmaString newString;

    DMA_CREATE_STRING(iMalloc->Alloc, wcslen(content), content, &newString)
    return newString;
}

void FreeDMAString(pDmaString pString)
{
    DMA_FREE_STRING(iMalloc->Free, pString)
    return;
}

// Procedure to retrieve content via docID to local file
// localFile is in URL form, e.g. "file:///\\temp\\foo.txt"
// Retrieves only the first content element of the first rendition

void MakeDocumentContentLocal(pDmaString documentID, wchar_t *localFile)
{
    pDmaString          localFileURL = NULL;
    IdmaProperties*      iDocVer      = NULL;
    IdmaListOfObject*    iRenditions  = NULL;
    IdmaListOfObject*    iContentEls  = NULL;
    IdmaContentTransfer* iFirstContent = NULL;
    IdmaProperties*      iFirstRendition = NULL;

    // Connect to the document version object. This makes a snapshot
    // of the persistent doc ver in the client's process space.
```

```
DmaRC rc = iDocSpace->ConnectObject (documentID, IID_IdmaProperties,
                                     (pDmapv)&iDocVer);

if (rc != DMARC_OK)
    Abort (rc, "Failed to connect to doc ver");

// NOTE: for brevity, further error checking is omitted.  All DMA calls
// should be followed by code similar to the above

// Get the first rendition
rc = iDocVer->GetPropValObjectById(&dmaprop_Renditions, IID_IdmaListOfObject,
                                  (pDmapv) &iRenditions);
rc = iRenditions->GetObject(0, IID_IdmaProperties, (pDmapv) &iFirstRendition);

// Get first content element
rc = iFirstRendition->GetPropValObjectById(&dmaprop_ContentElements,
                                           IID_IdmaListOfObject, (pDmapv) &iContentEls);
rc = iContentEls->GetObject(0, IID_IdmaContentTransfer, (pDmapv) &iFirstContent);

// From the content transfer element, get a file access object,
// and use it to copy the content to a local file
localFileURL = MakeDmaString(localFile);

rc = iFirstContent->CopyToResource (localFileURL, DMA_CREATE_REPLACE_EXISTING);

// cleanup
if (iRenditions != NULL) iRenditions->Release();
if (iFirstRendition != NULL) iFirstRendition->Release();
if (iContentEls != NULL) iContentEls->Release();
if (iFirstContent != NULL) iFirstContent->Release();
if (iDocVer != NULL) iDocVer->Release();
FreeDMAString(localFileURL);
}
```

5.3 Pretty Print Example

```
/*
 * Copyright 1995,1996,1997 The Association for Information and Image Manage-
ment Int'l
 *
 *                               1100 Wayne Avenue
 *                               Silver Spring, MD 20910-5603
 *                               Tel: 301/587-8202
 *                               Fax: 301/587-2711
 *
 *       All Rights Reserved.
 *       DMA (Document Management Alliance) working group.
 *
 *
 * Code example: Pretty printing of a DMA object's properties.
 *
 */

#include <stdio.h>

#include <dmatypes.h>
#include <dmacom.h>
#include <dmastr.h>
#include <dmarc.h>
#include <dmaiface.h>
#include <dmaenums.h>
#include <dmaids.h>
#include <dmaidvar.h>

/* Global reference to in effect IMalloc */
extern IMalloc *g_pMalloc;

char * g_rgpszTypes[] =
{
    "Illegal(0)",
    "Binary",
    "Boolean",
    "DateTime",
    "Float64",
    "Id",
    "Integer32",
    "Object",
    "String",
};

void PrintDmaId (
```

```
DmaId *pId )
{
    printf ( "%8.8lx-%4.4lx-%4.4lx-", pId->Data1, pId->Data2, pId->Data3 );
    printf ( "%2.2x%2.2x-", pId->Data4[0], pId->Data4[1] );
    for ( int I = 2; I < 8 ; I++ )
    {
        printf("%2.2x", pId->Data4[I] );
    }
}

void GetAndPrintPropByIndex (
    IdmaProperties *pprop,
    DmaIndex32 iProp,
    DmaInteger32 lDatatype )
{
    DmaRC rc;

    switch ( lDatatype )
    {
    case DMA_DATATYPE_BOOLEAN:
        {
            DmaBoolean bValue;
            rc = pprop->GetPropValBooleanByIndex ( iProp, &bValue );
            if ( rc == DMARC_OK )
            {
                printf ( "%s\n", bValue ? "TRUE" : "FALSE" );
            }
        }
        break;

    case DMA_DATATYPE_INTEGER32:
        {
            DmaInteger32 lValue;
            rc = pprop->GetPropValInteger32ByIndex ( iProp, &lValue );
            if ( rc == DMARC_OK )
            {
                printf ( "%d\n", lValue );
            }
        }
        break;

    case DMA_DATATYPE_FLOAT64:
        {
            DmaFloat64 fValue;
```

```
        rc = pprop->GetPropValFloat64ByIndex ( iProp, &fValue );
        if ( rc == DMARC_OK )
        {
            printf ( "%f\n", fValue );
        }
    }
    break;

case DMA_DATATYPE_STRING:
    {
        pDmaString pValue;
        rc = pprop->GetPropValStringByIndex ( iProp, &pValue );
        if ( rc == DMARC_OK )
        {
            printf ( "%ws\n", DMA_GET_STRING_TEXT(pValue) );
            DMA_FREE_STRING ( g_pMalloc->Free, pValue );
        }
    }
    break;

case DMA_DATATYPE_DATETIME:
    {
        pDmaString pValue;
        rc = pprop->GetPropValDateTimeByIndex ( iProp, &pValue );
        if ( rc == DMARC_OK )
        {
            printf ( "%ws\n", DMA_GET_STRING_TEXT(pValue) );
            DMA_FREE_STRING ( g_pMalloc->Free, pValue );
        }
    }
    break;

case DMA_DATATYPE_ID:
    {
        DmaId idValue;
        rc = pprop->GetPropValIdByIndex ( iProp, &idValue );
        if ( rc == DMARC_OK )
        {
            PrintDmaId ( &idValue );
            printf ( "\n" );
        }
    }
    break;
```

```
case DMA_DATATYPE_OBJECT:
{
    IdmaProperties *ppropSub;
    rc = pprop->GetPropValObjectByIndex ( iProp, IID_IdmaProperties,
                                           (pDmapv)&ppropSub );

    if ( rc == DMARC_OK )
    {
        printf ( "OBJECT\n" );

        ppropSub->Release();
    }
}
break;

case DMA_DATATYPE_BINARY:
{
    DmaBinaryValue dbvValue;
    rc = pprop->GetPropValBinaryByIndex ( iProp, &dbvValue );
    if ( rc == DMARC_OK )
    {
        DmaUInteger32 ulLineCount = 0;
        for ( DmaUInteger32 iByte = 0; iByte < dbvValue.cbBytes ; iByte++
)
        {
            printf ( "%2.2x", dbvValue.pbBytes[iByte] );
            if ( ulLineCount == 7 )
                printf ( "-" );
            else if ( ulLineCount == 15 )
                printf ( "\n" );
            else
                printf ( " " );

            ulLineCount = (ulLineCount + 1) % 16;
        }

        g_pMalloc->Free ( dbvValue.pbBytes );
    }
}
break;

default:
    printf ( "UNKNOWN DATATYPE %d\n", lDatatype );
    rc = DMARC_OK;
    break;
```



```
    }

    if ( rc == DMARC_VALUE_NOT_SET )
    {
        printf ( "VALUE NOT SET\n" );
    }
    else if ( rc != DMARC_OK )
    {
        printf ( "ERROR[%lx]\n", rc);
    }
}

void PrettyPrint (
    IdmaProperties *pObj )
{
    DmaRC rc;

    IdmaProperties *ppropCD;

    // First, get the class descriptor for the object
    rc = pObj->GetPropValObjectById ( &dmaProp_ClassDescription,
                                      IID_IdmaProperties,
                                      (pDmapv) &ppropCD );

    if ( rc == DMARC_OK )
    {
        // Further error handling omitted

        IdmaListOfObject *pPropList;
        rc = ppropCD->GetPropValObjectById ( &dmaProp_PropertyDescriptions,
                                              IID_IdmaListOfObject,
                                              (pDmapv) &pPropList );

        DmaInteger32 cProps;
        rc = pPropList->GetElementCount ( &cProps );

        for ( int iProp = 0; iProp < cProps ; iProp++ )
        {
            DmaRC rc;

            IdmaProperties *ppropPD;

            printf("%d: ", iProp );

            rc = pPropList->GetObject ( iProp, IID_IdmaProperties,
```

```
(pDmapv)&ppropPD );

pDmaString pName;

rc = ppropPD->GetPropValStringById ( &dmaProp_DisplayName,
                                     &pName );
printf ( "%ws: ", DMA_GET_STRING_TEXT(pName) );
DMA_FREE_STRING ( g_pMalloc->Free, pName );

DmaInteger32 lDatatype;
rc = ppropPD->GetPropValInteger32ById ( &dmaProp_DataType,
                                       &lDatatype );

DmaInteger32 lCardinality;
rc = ppropPD->GetPropValInteger32ById ( &dmaProp_Cardinality,
                                       &lCardinality );

if ( lCardinality == DMA_CARDINALITY_SINGLE )
{
    GetAndPrintPropByIndex ( pObj, iProp, lDatatype );
}
else if ( lCardinality == DMA_CARDINALITY_LIST )
{
    printf ( "LIST of %s: ", g_rgpszTypes[lDatatype] );
}
else
{
    printf ( "ENUM of %s\n", g_rgpszTypes[lDatatype] );
}

ppropPD->Release();
}

pPropList->Release();
ppropCD->Release();
}
else
{
    printf ( "Error %lx retrieving class description\n", rc );
}
}
```

5.4 Query Examples

Actual clients of DMA will most often implement queries as part of a user interface module. It is expected that the client user interface code will usually either take user interface gestures or a query language as input. However, it is not appropriate to show a user interface module or a query language parser as part of a DMA query example in this document: What needs to be illustrated is not user interface code or query language code, but the code that deals with DMA objects involved with DMA queries. Therefore, we have chosen a data driven style for the query example. It would be convenient for both clients and implementations of the DMA API to implement canned queries using the data driven style illustrated. One situation in which canned queries are useful in implementing the DMA API is in implementing navigation, e.g., finding the contents of a container, or the containers of a containee.

In our data driven style of example, there are several source files. One source file implements a general query parse tree generation engine that can generate a query parse tree for any query that the DMA query model can handle, with one exception: DMA query objects for list of constants (i.e., lists of integer constants, etc.) are omitted. Adding them would add bulk but to the example but not increase understanding. Adding lists of constants would be straightforward.

The parse tree generation engine has an include file that modules using the engine must include. The third and final source file is the module that is the client of the parse tree generation module. The client module has the data for a specific query.

The source files involved are:

- QUERY.H: This is the include file for query parse tree generation module. Both the query parse tree generation module and the client module include this file. This file includes all the DMA header files that are needed.
- QUERY.CPP: This is the query parse tree generation module. It is passed a tree of QueryNode structures (see below). It converts the tree of QueryNode structures into an equivalent DMA parse tree of DMA objects, and calls the ExecuteSearch method of the input DMA Scope object.
- QUEXP1.CPP: This is the client of the query parse tree generation module. It statically declares a tree of initialized QueryNode structures in and passes it to the parse tree generation module. The parse tree generation module returns a DMA Query Result Set object, which the client uses to enumerate the results of the query.
- QUEXP2.CPP: This example demonstrates how containment can be expressed within the DMA query model.
- QUEXP3.CPP: This example demonstrates how children of an object supporting direct containment can be navigated.

5.4.1 query.h

Query parse trees in the DMA query model consist of DMA objects that merely contain data, and do not have any interesting methods except those dealing with properties. (The `IdmaProperties` and `IdmaEditProperties` interfaces deal with properties.) The "query.h" file has a "typedef struct" declaration for a collection of C structures that can hold the information for any node in a DMA query parse tree. These structures are `Operator`, `BinaryConst`, `StringConst`, `BooleanConst`, `IntegerConst`, `FloatConst`, `DateTimeConst`, `IDConst`, `OIIDConst`, `Property`, `FromExprElt`, `JoinExprElt`, `OrderElt`, `QueryNode`, and `QueryRoot`. The `QueryNode` structure contains a union field that has a case for each of the other structs.

The basic idea is that the client statically declares an array of `QueryNode` structures that represents a depth-first tree of initialized `QueryNode` structures equivalent to a DMA query parse tree. This file also contains the procedure signature of the "entry point" into the parse tree module. Finally, this file includes all the DMA header files that are necessary.

5.4.2 query.cpp

This source file is the query parse tree generation module. The entry point function is `StartQuery`. It takes a `QueryRoot` structure and a DMA Scope object as input. The `QueryRoot` structure is the topmost node in a tree of structures for the main query. Subqueries are allowed. In that case, one of the subordinate nodes will also be a `QueryRoot` structure.

One of the functions `StartQuery` calls is `BuildQueryExpr`. This function is indirectly recursive – it calls `BuildTree`, which can call `BuildQueryExpr`. The function `BuildTree` is also directly recursive – it can call itself.

The main query, and each subquery (if any) are represented by separate trees of C structures. The "level" field of the `QueryNode` structure represents the recursive level of the node in the tree. The value of "level" for the root node is zero. The trees are flattened into an array of `QueryNode` structures in which the query nodes occur in depth first order.

5.4.3 quexp1.cpp

This file declares the particular query to be executed and calls `StartSearch` in the parse tree generation module. It uses the Query Result Set object returned to enumerate the result rows of the query.

5.4.4 QUERY.H Example File

```
/* query.h
**
**          PRODUCE QUERIES FROM SIMPLE DATA STRUCTURES
**
**/

/*
 * Copyright 1995, 1996, 1997, 1998 by the
 * Association for Information and Image Management International
 * 1100 Wayne Avenue
 * Silver Spring, MD 20910-5603
 * Tel: 301/587-8202
 * Fax: 301/587-2711
 * All Rights Reserved.
 * DMA (Document Management Alliance) working group.
 */

/* The function
**
**          DmaRC StartQuery(          QueryRoot *pQR,
**                                IdmaScope *pScope,
**                                IdmaResultSet **ppResultSet )
**
** takes the QueryRoot datastructure and uses the *pScope scope object
** to construct a full DMA query, submit it for execution, and return
** the Result set delivered by the query request.
**
** If successful, StartQuery returns the result set interface of the
** result set that is delivered from the successful query initiation.
**
** If unsuccessful, an appropriate failure result code is delivered and
** no result set is available.
**
** The QueryRoot data type, and all of those needed to support it, are
** defined here. They specify how queries can be populated from "canned"
** memory structures.
**/

#include <dmatypes.h>

typedef struct Operator
{
    DmaId *          pID;
}
    Operator;

typedef struct DmaBinaryValue    BinaryConst;

typedef struct StringConst
{
    pDmaString          val;
}
    StringConst;

typedef struct BooleanConst
```

```
{
    DmaBoolean          val;
}
BooleanConst;

typedef struct IntegerConst
{
    DmaInteger32        val;
}
IntegerConst;

typedef struct FloatConst
{
    DmaFloat64          val;
}
FloatConst;

typedef struct DateTimeConst
{
    pDmaDateTime        val;
}
DateTimeConst;

typedef struct IDConst
{
    DmaId *              pID;
}
IDConst;

typedef struct Property
{
    DmaInteger32         occurx;
    DmaId *              pPropID;
}
Property;

/* The Selections list is represented by an array of Property.

   The last array element is used only as a stopper entry. This is
   indicated by occurx = -1.
*/

/* A From Expression is represented by an array of FromExprElt.

   The first array element must have valid values only for the
   occurx and pClassID fields. The other fields are ignored.

   The intermediate array elements must have all fields valid (except
   for cross joins, which ignores pEqualOp, occurx1, pPropID1,
   occurx2, pPropID2).

   The last array element is used only as a stopper element. This
   is indicated by occurx = -1.
*/
typedef struct FromExprElt
{
    DmaInteger32         occurx;    /* Searchable class occurrence index */
    DmaId *              pClassID; /* ID of searchable class */
    DmaBoolean           InclSubclasses;
```

```
DmaId *                pJoinID; /* ID of the join operator */

DmaId *                pEqualOp; /* type dependent equality operator */

DmaInteger32          occurx1; /* searchable class occurrence */
DmaId *               pPropID1; /* property ID */

DmaInteger32          occurx2; /* searchable class occurrence */
DmaId *               pPropID2; /* property ID */
}

JoinExprElt;

/* The Orderings list is represented by an array of OrderElt.

The last array element is used only as a stopper entry. This
is indicated by selectionx = -1.
*/
typedef struct OrderElt
{
    DmaInteger32          selectionx; /* -1 if stopper entry */
    DmaBoolean            descending;
}

OrderElt;

typedef struct QueryNode
{
    DmaInteger32          type; /* type of query node */
#   define StopperNodeType      0
#   define OperatorNodeType    1
#   define BinaryConstNodeType 2
#   define StringConstNodeType 3
#   define BooleanConstNodeType 4
#   define IntegerConstNodeType 5
#   define FloatConstNodeType 6
#   define DateTimeConstNodeType 7
#   define IDConstNodeType     8
#   define PropertyNodeType    9
#   define QueryRootNodeType   10

    DmaInteger32          level; /* recursive level of node. Starts at 0.
                                -1 for stopper node.
                                */

    union
    {
        void *            pVoid;
        Operator *         pOperator;
        BinaryConst *      pBinary;
        StringConst *      pString;
        BooleanConst *      pBoolean;
        IntegerConst *      pInteger;
        FloatConst *        pFloat;
        DateTimeConst *     pDateTime;
        IDConst *           pID;
        Property *          pProperty;
        struct QueryRoot *  pQueryRoot;
    }

    u;
}

QueryNode;
```

```
typedef struct QueryRoot
{
    JoinExprElt *      pFromExpr;
    Property *         pSelections;
    QueryNode *        pQueryExpr;
    OrderElt *         pOrder;
    DmaBoolean          distinct;
}
    QueryRoot;

/* External module entry points */

extern DmaRC
StartQuery(
    QueryRoot *      pQR,
    IdmaScope *      pScope,
    IdmaResultSet ** ppResultSet);

/* end of query.h */
```

5.4.5 QUERY.CPP Example File

```
/* query.cpp
**
**          START A QUERY BASED ON A STORED IN-MEMORY QUERY TREE
**
*/

/*
* Copyright 1995, 1996, 1997, 1998 by the
* Association for Information and Image Management International
* 1100 Wayne Avenue
* Silver Spring, MD 20910-5603
* Tel: 301/587-8202
* Fax: 301/587-2711
* All Rights Reserved.
* DMA (Document Management Alliance) working group.
*/

/* The function
**
**          DmaRC StartQuery(          QueryRoot *pQR,
**                                IdmaScope *pScope,
**                                IdmaResultSet **ppResultSet )
**
** takes the QueryRoot datastructure and uses the *pScope scope object
** to construct a full DMA query, submit it for execution, and return
** the Result set delivered by the query request.
**
** If successful, StartQuery returns the result set interface of the
** result set that is delivered from the successful query initiation.
**
** If unsuccessful, an appropriate failure result code is delivered and
** no result set is provided.
```



```
*/

/* Limitations of this sample:
**
**      1. Memory leaks (e.g., pNode) can occur in BuildTree(). The "theend"
**         branch should provide cleanup.
**      2. This procedure is technically not thread safe. It depends on access
**         to a global array, CD[], which is used for a cache.
**
**
*/

#include <stddef.h>
#include <dmaiface.h>
#include <dmarc.h>
#include <dmaids.h>
#include <dmaidvar.h>

#include "query.h"

typedef struct CDElt
{
    DmaId *          pID;
    IdmaClassDescription * pCD;
}

CDElt;

/* Class Descriptions objects for query classes.
These are used to create QueryNode objects during query construction.
The DmaId of the desired class is looked up in CD [],
and the corresponding IdmaClassDescription::New() method is used
to create a fresh QueryNode object of the corresponding class.

The initialization of the pCD fields is done dynamically at run time
the first time a query is executed. This is accomplished by traversing the
QueryMetadataClasses property of the Scope object.
*/

static CDElt CD [] =
{
    { &dmaClass_QueryRoot, NULL },
    { &dmaClass_QueryProperty, NULL },
    { &dmaClass_QueryOrderByNode, NULL },
    { &dmaClass_QueryConstantBinary, NULL },
    { &dmaClass_QueryConstantString, NULL },
    { &dmaClass_QueryConstantBoolean, NULL },
    { &dmaClass_QueryConstantInteger32, NULL },
    { &dmaClass_QueryConstantFloat64, NULL },
    { &dmaClass_QueryConstantDateTime, NULL },
    { &dmaClass_QueryConstantId, NULL },
    { &dmaClass_QuerySearchableClass, NULL },
    { &dmaClass_QueryJoinOperator, NULL },
    { &dmaClass_QueryOperator, NULL },
    { &dmaClass_QueryResultSet, NULL },
    { &dmaClass_QueryResultRow, NULL },
    { &dmaClass_ListOfObject, NULL },
    { NULL, NULL }
};
```

```
/*
    All the query examples follow the following conventions:

    (1) All procedures have a label, "theend", that labels code
    prepared to handle all the cleanup after any error encountered
    by the procedure, and can handle the success case as well.

    (2) All procedures exit through the code labeled by "theend",
    regardless of success or failure.

    (3) All non null interface pointers are valid. Therefore,
    interface pointers are initialized to null in their declaration
    and conditionally released after "theend" label has been reached
    using the CHK_RELEASE macro.
*/

/* The CHK macro checks method return codes. It relies on the
   convention that the label "theend" always exists.
*/
#define CHK(rc_) \
{ \
    if ((rc_) != DMARC_OK) \
    { \
        goto theend; \
    } \
}

/* This macro adheres to the convention that non null interface
   pointers are valid.
*/
#define RELEASE(p_)      { p_>Release(); p_ = NULL; }

/* This macro conditionally releases an interface pointer.

   This macro adheres to the convention that non null interface
   pointers are valid.
*/
#define CHK_RELEASE(p_)  { if (p_ != NULL) RELEASE(p_); }

/* This procedure tests two DmaId's for equality.
*/
static DmaBoolean
IdEqual(
    DmaId *          pID1,
    DmaId *          pID2)
{
    DmaUInteger32 *   pX = (DmaUInteger32 *)pID1;
    DmaUInteger32 *   pY = (DmaUInteger32 *)pID2;

    return (DmaBoolean)(pX[0] == pY[0] && pX[1] == pY[1] &&
        pX[2] == pY[2] && pX[3] == pY[3]);
}

/* This procedure is called to finish initializing the CD []
   global array. It sets the CD[i].pCD field to the interface
   pointer to the appropriate class description object.
```

One reason Class Description objects are useful is that you can create a fresh instance of the class they describe by calling their `IdmaObjectFactory::CreateObject()` method.

```
*/
static DmaRC
InitQueryCDOObjects(
    IdmaScope *          pScope)
{
    DmaRC                rc;
    IdmaProperties *      pProp = NULL;
    IdmaListOfObject *    pQNodes = NULL;
    IdmaClassDescription * pQNode = NULL;
    IdmaProperties *      pCD = NULL;
    IdmaListOfId *        pIDs = NULL;
    DmaId                 ID;
    DmaIndex32            count;
    DmaIndex32            IdCount;
    DmaIndex32            i;
    DmaIndex32            j;
    DmaIndex32            k;

    rc = pScope->QueryInterface(IID_IdmaProperties, (pDmapv)(&pProp));
    CHK(rc);

    rc = pProp->GetPropValObjectById(
        &dmaProp_QueryConstructionClassDescriptions,
        IID_IdmaListOfObject,
        (pDmapv)&pQNodes);

    CHK(rc);

    rc = pQNodes->GetElementCount(&count);
    CHK(rc);

    i = 0;
    while (i < count)
    {
        rc = pQNodes->GetObject(i, IID_IdmaProperties, (pDmapv)&pCD);
        CHK(rc);

        j = 0;
        while (CD[j].pID != NULL)
        {
            rc = pCD->GetPropValObjectById(&dmaProp_Ids,
                                           IID_IdmaListOfId,
                                           (pDmapv)(&pIDs));

            CHK(rc);

            rc = pIDs->GetElementCount(&IdCount);
            CHK(rc);

            k = 0;
            while (k < IdCount)
            {
                rc = pIDs->GetId(k, &ID);
                CHK(rc);
                if (IdEqual(&ID, CD[j].pID))
                {
                    rc = pCD->QueryInterface(IID_IdmaClassDescription,
                                             (pDmapv)(&CD[j].pCD));
                }
            }
        }
        i++;
    }
}
```

```
        CHK(rc);
        break;
    }

    ++k;
} /* while */

RELEASE(pIDs);

    ++j;
} /* while */

RELEASE(pCD);
    ++i;
} /* while */

rc = DMARC_OK;

theend:
    CHK_RELEASE(pIDs);
    CHK_RELEASE(pProp);
    CHK_RELEASE(pQNodes);
    CHK_RELEASE(pQNode);
    CHK_RELEASE(pCD);

    return rc;
}

/* This procedure returns a fresh query object of class *pClassId
   in *ppNew.
*/
static DmaRC
FreshQueryObject(
    IdmaScope *          pScope,
    DmaId *              pClassID,
    IdmaEditProperties ** ppNew)
{
    DmaRC          rc;
    DmaInteger32    i;
    IdmaObjectFactory * pFact = NULL;

    *ppNew = NULL;

    if (CD[0].pID == NULL)
    {
        rc = InitQueryCDOObjects(pScope);
        CHK(rc);
    }

    /* Look up the class by its ID in the CD[] array.
       Then call the IdmaObjectFactory::CreateObject() method
       on the appropriate class description object.
    */
    i = 0;
    while (CD[i].pID != NULL)
    {
        if (IdEqual(CD[i].pID, pClassID))
        {
            rc = CD[i].pCD->QueryInterface(IID_IdmaObjectFactory,
```

```
(pDmapv)&pFact);

CHK(rc);
rc = pFact->CreateObject(pClassID,
                        IID_IdmaEditProperties,
                        (pDmapv)ppNew);

CHK(rc);
RELEASE(pFact);
rc = DMARC_OK;
goto theend;
}
++i;
}

rc = DMARC_BAD_PARAMETER;

theend:
return rc;
}

/* This procedure establishes the FromExpression of a
   QueryRoot object.
*/
static DmaRC
SetFromExpr(
    IdmaScope *          pScope,
    IdmaEditProperties * pQueryRoot,
    FromExprElt *        pFromExpr)
{
    DmaRC rc;
    IdmaEditProperties * pSC;
    IdmaEditProperties * pRslt = NULL;
    IdmaEditProperties * pJoinOp = NULL;
    IdmaEditListOfObject * pOperands = NULL;
    IdmaEditProperties * pEqLOp = NULL;
    IdmaEditListOfObject * pEqLOperands = NULL;
    IdmaEditProperties * pOpnd = NULL;

    /* Deal with the first searchable class as a special case. */
    rc = FreshQueryObject(pScope, &dmaClass_QuerySearchableClass, &pJoinOp);
    CHK(rc);
    rc = pJoinOp->PutPropValIdById(&dmaProp_SearchableClassId,
                                   pFromExpr->pClassID);

    CHK(rc);
    rc = pJoinOp->PutPropValInteger32ById(&dmaProp_SearchableClassOccurrence,
                                           pFromExpr->occurx);
    pRslt = pJoinOp; pJoinOp = NULL; /* in case there are no joins */

    ++pFromExpr; /* advance to list element 1 */

    /* deal with the next join operator. Join to previous result. */
    while (pFromExpr->occurx >= 0)
    {
        pRslt->Release(); /* We are done with the previous result */

        /* Get a fresh Join Operator object */
        rc = FreshQueryObject(pScope, &dmaClass_QueryJoinOperator,
                               &pJoinOp);

        CHK(rc);
```

```
/* Get list of operands of the Join operator object */
rc = pJoinOp->GetPropValObjectById(&dmaProp_Operands,
                                   IID_IdmaEditListOfObject,
                                   (pDmapv)&pOperands);

CHK(rc);

/* Set operand[0] of Join operator object */
rc = pOperands->InsertObject(0, (Dmapv)pRslt);
CHK(rc);
RELEASE(pRslt);

/* Set operand[1] of Join operator object */
rc = FreshQueryObject(pScope, &dmaClass_QuerySearchableClass, &pSC);
CHK(rc);
rc = pSC->PutPropValIdById(&dmaProp_SearchableClassId,
                          pFromExpr->pClassID);

CHK(rc);
rc = pSC->PutPropValInteger32ById(
    &dmaProp_SearchableClassOccurrence,
    pFromExpr->occurx);
rc = pOperands->InsertObject(1, (Dmapv)pSC);
CHK(rc);
RELEASE(pSC);

/* Get Equal operator object */
rc = FreshQueryObject(pScope, &dmaClass_QueryOperator, &pEqOp);
CHK(rc);
rc = pEqOp->PutPropValIdById(&dmaProp_QueryOperatorId,
                             pFromExpr->pEqualOp);

/* Get list of operands of Equal operator object */
rc = pEqOp->GetPropValObjectById(&dmaProp_Operands,
                                 IID_IdmaEditListOfObject,
                                 (pDmapv)&pEqOperands);

/* set operand[0] of Equal operaor object */
rc = FreshQueryObject(pScope, &dmaClass_QueryProperty, &pOpnd);
CHK(rc);
rc = pOpnd->PutPropValInteger32ById(
    &dmaProp_SearchableClassOccurrence,
    pFromExpr->occurx1);

CHK(rc);
rc = pOpnd->PutPropValIdById(&dmaProp_PropertyId,
                             pFromExpr->pPropID1);

CHK(rc);
rc = pEqOperands->InsertObject(0, (Dmapv)pOpnd);
RELEASE(pOpnd);

/* Set operand[1] of Equal operator object */
rc = FreshQueryObject(pScope, &dmaClass_QueryProperty, &pOpnd);
CHK(rc);
rc = pOpnd->PutPropValInteger32ById(
    &dmaProp_SearchableClassOccurrence,
    pFromExpr->occurx2);

CHK(rc);
rc = pOpnd->PutPropValIdById(&dmaProp_PropertyId,
                             pFromExpr->pPropID2);

CHK(rc);
rc = pEqOperands->InsertObject(1, (Dmapv)pOpnd);
RELEASE(pOpnd);
```

```
/* We are now done with the operands of the Equal operator */
RELEASE(pEqlOperands);

/* Set operand[2] of Join operator object */
rc = pOperands->InsertObject(2, (Dmapv)pEqlOp);
CHK(rc);

/* We are now done with the Equal operator object */
RELEASE(pEqlOp);

/* We are now done with the operands of the join operator */
RELEASE(pOperands);

/* advance to next join operation */
pRslt = pJoinOp; pJoinOp = NULL;
++pFromExpr;
} /* while */

/* Set the FromExpr property in the Query Root object */
rc = pQueryRoot->PutPropValObjectById(&dmaProp_FromExpression,
                                     (Dmapv)pRslt);

CHK(rc);
RELEASE(pRslt);

rc = DMARC_OK;

theend:
CHK_RELEASE(pRslt);
CHK_RELEASE(pSC);
CHK_RELEASE(pJoinOp);
CHK_RELEASE(pOperands);
CHK_RELEASE(pEqlOp);
CHK_RELEASE(pEqlOperands);
CHK_RELEASE(pOpnd);

return rc;
}

/* This procedure establishes the Selections property of a
QueryRoot object.
*/
static DmaRC
SetSelections(
    IdmaScope *          pScope,
    IdmaEditProperties *  pQueryRoot,
    Property *           pSelections)
{
    DmaRC          rc;
    DmaInteger32    i;
    IdmaEditProperties * pNode = NULL;
    IdmaEditListOfObject * pA = NULL;

    if (pSelections == NULL)
    {
        return DMARC_OK;
    }

    rc = pQueryRoot->GetPropValObjectById(&dmaProp_Selection,
```

```
IID_IdmaEditListOfObject,
(pDmapv)&pA);

CHK(rc);

i = 0;
while (pSelections->occurx >= 0)
{
    rc = FreshQueryObject(pScope, &dmaClass_QueryProperty, &pNode);
    CHK(rc);
    rc = pNode->PutPropValInteger32ById(
        &dmaProp_SearchableClassOccurrence,
        pSelections->occurx);

    CHK(rc);
    rc = pNode->PutPropValIdById(&dmaProp_PropertyId,
        pSelections->pPropID);

    CHK(rc);

    rc = pA->InsertObject(i, (Dmapv)pNode);
    CHK(rc);
    RELEASE(pNode);

    ++i; ++pSelections;
}

rc = DMARC_OK;

theend:
CHK_RELEASE(pNode)
CHK_RELEASE(pA);

return rc;
}

/* This procedure establishes the Orderings property of a
QueryRoot object.
*/
static DmaRC
SetOrderings(
    IdmaScope *      pScope,
    IdmaEditProperties * pQueryRoot,
    OrderElt *      pOrder)
{
    DmaRC          rc;
    DmaInteger32   i;
    IdmaEditProperties * pNode = NULL;
    IdmaEditProperties * pX = NULL;
    IdmaEditListOfObject * pA = NULL;

    if (pOrder == NULL)
    {
        return DMARC_OK;
    }

    rc = pQueryRoot ->GetPropValObjectById
        (
            &dmaProp_Orderings,
            IID_IdmaEditListOfObject,
            (pDmapv)&pA );

    CHK(rc);
```



```
i = 0;
while (pOrder->selectionsx >= 0)
{
    rc = FreshQueryObject(pScope, &dmaClass_QueryOrderByNode, &pNode);
    CHK(rc);
    rc = pNode->PutPropValInteger32ById(&dmaProp_SelectionsIndex,
                                       pOrder->selectionsx);

    CHK(rc);
    rc = pNode->PutPropValBooleanById(&dmaProp_DescendingRequested,
                                       pOrder->descending);

    CHK(rc);

    rc = pA->InsertObject(i, (Dmapv)pNode);
    CHK(rc);

    RELEASE(pNode);

    ++i; ++pOrder;
}

rc = DMARC_OK;

theend:
    CHK_RELEASE(pNode);
    CHK_RELEASE(pA);

    return rc;
}

static DmaRC
BuildTree(
    IdmaScope *          pScope,
    QueryNode **         ppQueryExpr,
    IdmaEditProperties ** ppNode);
    // Provide a declarator for BuildTree to satisfy BuildQueryExpr().

/* This procedure builds the QueryExpression property of a
   QueryRoot object. It is mutually-recursive with BuildTree().
*/
static DmaRC
BuildQueryExpr(
    IdmaScope *          pScope,
    QueryNode *          pQueryExpr,
    IdmaEditProperties ** ppQE)
{
    DmaRC                rc;

    *ppQE = NULL;
    rc = BuildTree(pScope, &pQueryExpr, ppQE);
    return rc;
}

/* This is a recursive procedure to build a query expression tree.

The nodes in the query nodes input array (**ppQueryExpr)
occur in depth first order.

The procedure advances *ppQueryExpr by one element. When
the recursion fully unwinds, the pointer points at the stopper
```

element.

The procedure returns the DMA object it develops in *ppNode at each recursive level.

```
*/
static DmaRC
BuildTree(
    IdmaScope *          pScope,
    QueryNode **         ppQueryExpr,
    IdmaEditProperties ** ppNode)
{
    DmaRC rc;
    QueryNode * pQueryExpr = *ppQueryExpr;
    DmaInteger32 level = pQueryExpr->level;
    IdmaEditProperties * pNode = NULL;
    IdmaEditProperties * pOpnd = NULL;
    IdmaEditListOfObject * pOperands = NULL;
    IdmaListOfObject * pOpndList = NULL;
    IdmaEditProperties * pOpndProps = NULL;
    DmaIndex32 count;

    *ppNode = NULL;

    switch (pQueryExpr->type)
    {
    case StopperNodeType:
        /* Actually, if we get here, that's a bug. */
        break;

    case OperatorNodeType:
        rc = FreshQueryObject(pScope, &dmaClass_QueryOperator, &pNode);
        CHK(rc);
        rc = pNode->PutPropValIdById(&dmaProp_QueryOperatorId,
                                    pQueryExpr->u.pOperator->pID);
        *ppQueryExpr = ++pQueryExpr;
        while (pQueryExpr->level == level + 1) /* develop the operands */
        {
            rc = BuildTree(pScope, ppQueryExpr, &pOpnd); /* RECURSE */
            CHK(rc);
            rc = pNode->GetPropValObjectById(&dmaProp_Operands,
                                            IID_IdmaEditListOfObject,
                                            (pDmapv)&pOperands);

            if (rc != DMARC_OK)
            {
                goto theend;
            }

            rc = pOperands->QueryInterface(IID_IdmaListOfObject,
                                           (pDmapv)&pOpndList);
            CHK(rc);

            rc = pOpndList->GetElementCount(&count);
            CHK(rc);

            rc = pOperands->InsertObject(count, pOpnd);
            CHK(rc);

            RELEASE(pOpnd);
            RELEASE(pOperands);
            RELEASE(pOpndList);
        }
    }
}
```

```
        pQueryExpr = *ppQueryExpr;
    }
    *ppNode = pNode;
    break;

case BinaryConstNodeType:
    rc = FreshQueryObject(pScope, &dmaClass_QueryConstantBinary,
                          &pNode);

    CHK(rc);
    rc = pNode->PutPropValBinaryById(&dmaProp_ConstantValueBinary,
                                     pQueryExpr->u.pBinary);

    CHK(rc);
    *ppQueryExpr = ++pQueryExpr;
    *ppNode = pNode;
    break;

case StringConstNodeType:
    rc = FreshQueryObject(pScope, &dmaClass_QueryConstantString,
                          &pNode);

    CHK(rc);
    rc = pNode->PutPropValStringById(&dmaProp_ConstantValueString,
                                     pQueryExpr->u.pString->val);

    CHK(rc);
    *ppQueryExpr = ++pQueryExpr;
    *ppNode = pNode;
    break;

case BooleanConstNodeType:
    rc = FreshQueryObject(pScope, &dmaClass_QueryConstantBoolean,
                          &pNode);

    CHK(rc);
    rc = pNode->PutPropValBooleanById(&dmaProp_ConstantValueBoolean,
                                     pQueryExpr->u.pBoolean->val);

    CHK(rc);
    *ppQueryExpr = ++pQueryExpr;
    *ppNode = pNode;
    break;

case IntegerConstNodeType:
    rc = FreshQueryObject(pScope, &dmaClass_QueryConstantInteger32,
                          &pNode);

    CHK(rc);
    rc = pNode->PutPropValInteger32ById(
        &dmaProp_ConstantValueInteger32,
        pQueryExpr->u.pInteger->val);

    CHK(rc);
    *ppQueryExpr = ++pQueryExpr;
    *ppNode = pNode;
    break;

case FloatConstNodeType:
    rc = FreshQueryObject(pScope, &dmaClass_QueryConstantFloat64,
                          &pNode);

    CHK(rc);
    rc = pNode->PutPropValFloat64ById(&dmaProp_ConstantValueFloat64,
                                     pQueryExpr->u.pFloat->val);

    CHK(rc);
    *ppQueryExpr = ++pQueryExpr;
    *ppNode = pNode;
```

```
        break;

    case DateTimeConstNodeType:
        rc = FreshQueryObject(pScope, &dmaClass_QueryConstantDateTime,
                               &pNode);

        CHK(rc);
        rc = pNode->PutPropValDateTimeById(&dmaProp_ConstantValueDateTime,
                                             pQueryExpr->u.pDateTime->val);

        CHK(rc);
        *ppQueryExpr = ++pQueryExpr;
        *ppNode = pNode;
        break;

    case IDConstNodeType:
        rc = FreshQueryObject(pScope, &dmaClass_QueryConstantId,
                               &pNode);

        CHK(rc);
        rc = pNode->PutPropValIdById(&dmaProp_ConstantValueId,
                                     pQueryExpr->u.pID->pID);

        CHK(rc);
        *ppQueryExpr = ++pQueryExpr;
        *ppNode = pNode;
        break;

    case PropertyNodeType:
        rc = FreshQueryObject(pScope, &dmaClass_QueryProperty,
                               &pNode);

        CHK(rc);
        rc = pNode->PutPropValInteger32ById(
            &dmaProp_SearchableClassOccurrence,
            pQueryExpr->u.pProperty->occurex);

        CHK(rc);
        rc = pNode->PutPropValIdById(&dmaProp_PropertyId,
                                     pQueryExpr->u.pProperty->pPropID);

        CHK(rc);
        *ppQueryExpr = ++pQueryExpr;
        *ppNode = pNode;
        break;

    case QueryRootNodeType:
        rc = BuildQueryExpr(pScope, pQueryExpr, ppNode);
        CHK(rc);
        *ppQueryExpr = ++pQueryExpr;
        break;

    default:
        rc = DMARC_BAD_PARAMETER;
        goto theend;
}

rc = DMARC_OK;

theend:
    return rc;
}

/* StartQuery() is the (only) external entry point of this module.
   All prior functions in this source file are only called
   internally in this source file.
```

StartQuery() develops a DMA Query Object and all its dependent Query Node objects, and then calls ExecuteSearch on the input Scope object. Then, it deallocates all the Query Node objects.

StartQuery() returns a pointer to the Query object in error (or NULL, if no error) in *ppErrnode, and it returns the Query Result Set object in *ppResultSet.

```
*/
DmaRC
StartQuery(
    QueryRoot *          pQR,
    IdmaScope *          pScope,
    IdmaResultSet **     ppResultSet)
{
    DmaRC                rc;
    IdmaEditProperties *  pQE = NULL;
    IdmaEditProperties *  pQueryObj = NULL;
    IdmaEditProperties *  pQueryRoot = NULL;

    *ppResultSet = NULL;

    /* Set pQueryObj = IdmaEditProperties interface of a fresh
       query object.
    */
    rc = FreshQueryObject(pScope, &dmaClass_Query, &pQueryObj);
    CHK(rc);

    /* Set pQueryRoot to IdmaEditProperties interface of a fresh query
       root object.
    */
    rc = FreshQueryObject(pScope, &dmaClass_QueryRoot, &pQueryRoot);
    CHK(rc);

    rc = SetFromExpr(pScope, pQueryRoot, pQR->pFromExpr);
    CHK(rc);

    rc = SetSelections(pScope, pQueryRoot, pQR->pSelections);
    CHK(rc);

    rc = SetOrderings(pScope, pQueryRoot, pQR->pOrder);
    CHK(rc);

    rc = BuildQueryExpr(pScope, pQR->pQueryExpr, &pQE);
    CHK(rc);

    rc = pQueryRoot->PutPropValObjectById(&dmaProp_QueryExpression,
                                           (Dmapv)pQE);
    CHK(rc);

    RELEASE(pQE);

    rc = pQueryRoot->PutPropValBooleanById(&dmaProp_DistinctRowsRequested,
                                           pQR->distinct);
    CHK(rc);

    pQueryObj->PutPropValObjectById(&dmaProp_QueryRoot,
                                    (Dmapv)pQueryRoot);
    CHK(rc);
}
```

```
RELEASE(pQueryRoot);

rc = pScope->ExecuteSearch((Dmapv)pQueryObj,
                           NULL,
                           DMA_FALSE,
                           IID_IdmaResultSet,
                           (pDmapv)ppResultSet);

CHK(rc);

rc = DMARC_OK;

theend:
CHK_RELEASE(pQE);
CHK_RELEASE(pQueryObj);
CHK_RELEASE(pQueryRoot);

return rc;
}

/* end of query.cpp */
```

5.4.6 QUERY1.CPP Example File

```
/*
 * Copyright 1995, 1996, 1997 by the
 * Association for Information and Image Management International
 * 1100 Wayne Avenue
 * Silver Spring, MD 20910-5603
 * Tel: 301/587-8202
 * Fax: 301/587-2711
 * All Rights Reserved.
 * DMA (Document Management Alliance) working group.
 */

/* quexpl.cpp

This example is for the following query:

SELECT S.s1, T.t1
FROM S INNER JOIN T ON S.s1 = T.t1
WHERE S.s2 > 3 AND T.t2 = "red"
ORDER BY <null>
*/

#define DMA_INIT_CD
#include <query.h>

/* Declare DmaId variables for the searchable classes.
```

```
*/
#define SVal { 0x12345678, 0x1234, 0x5678, \
              { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 } }

DMA_EXTERN_C DmaId S = SVal;

#define TVal { 0x12345678, 0x1234, 0x5678, \
              { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x09 } }

DMA_EXTERN_C DmaId T = TVal;

/* Declare DmaId variables for the properties.
*/
#define s1Val { 0x12345678, 0x1234, 0x5678, \
               { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x0A } }

DMA_EXTERN_C DmaId s1 = s1Val;

#define s2Val { 0x12345678, 0x1234, 0x5678, \
               { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x0B } }

DMA_EXTERN_C DmaId s2 = s2Val;

#define t1Val { 0x12345678, 0x1234, 0x5678, \
               { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x0C } }

DMA_EXTERN_C DmaId t1 = t1Val;

#define t2Val { 0x12345678, 0x1234, 0x5678, \
               { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x0D } }

DMA_EXTERN_C DmaId t2 = t2Val;

/* Declare the variables needed to perform the query.
*/

JoinExprElt          From [] =
{
    { 0, &S, DMA_FALSE, NULL, NULL, 0, NULL, 0, NULL },
    { 1, &T, DMA_FALSE, &dmaJoinOperator_Inner,
      &dmaQueryOperator_EqualInteger32, 0,&s1, 1, &t1 },
    { -1, NULL, DMA_FALSE, NULL, NULL, 0, NULL, 0, NULL }
};
```

```
Property          Select [] =
{
    { 0, &s1 },
    { 1, &t1 },
    { -1, NULL }
};

Operator          N1 = { &dmaQueryOperator_And };
Operator          N2 = { &dmaQueryOperator_GreaterInteger32 };
Property          N3 = { 0, &s2 };
IntegerConst      N4 = { 3 };
Operator          N5 = { &dmaQueryOperator_EqualString };
Property          N6 = { 1, &t2 };
StringConst       N7 = { L"red" };

QueryNode         QueryExpr [] =
{
    { OperatorNodeType, 0, &N1 },
    { OperatorNodeType, 1, &N2 },
    { PropertyNodeType, 2, &N3 },
    { IntegerConstNodeType, 2, &N4 },
    { OperatorNodeType, 1, &N5 },
    { PropertyNodeType, 2, &N6 },
    { StringConstNodeType, 2, &N7 },
    { StopperNodeType, -1, NULL }
};

QueryRoot         QR = { &From[0], &Select[0], &QueryExpr[0],
                        NULL, DMA_FALSE };

DmaRC
QueryExample1(void)
{
    DmaRC          rc;
    IdmaScope *    pScope = NULL;
    IdmaResultSet * pResultSet = NULL;
    IdmaProperties * pResultRow = NULL;
    DmaIndex32     SelectionsOffset;

    /* TBD: Not shown:
       Initializataion to get system object, doc space object,
       and target scope object = pScope.
```



```
*/

rc = StartQuery(&QR, pScope, &pResultSet);
CHK(rc);

while (1)
{
    rc = pResultSet->GetNextResultRow(IID_IdmaProperties,
                                      (pDmapv)&pResultRow);

    if (rc != DMARC_OK)
    {
        break;
    }

    rc = pResultRow->GetPropValInteger32ById(
        &dmaprop_SelectListOffset,
        &SelectionsOffset);

    CHK(rc);

    /* TBD: Not shown: Access the values of the properties of
       the query result row referred to by pResultRow
       corresponding to the properties selected in the Selections
       list, e.g.,

           DmaIndex32    x;
           <datatype>    y;
           rc = pResultRow->GetPropVal<datatype>ByIndex(
               SelectionsOffset + x,
               &y);

       to store the value for selection list element x into y.
    */

    RELEASE(pResultRow); /* Release the Query Result row */
}

theend:
return(DMARC_OK);
}
```

5.4.7 QUERY2.CPP Example File

```
/*
 * Copyright 1995, 1996, 1997 by the
 * Association for Information and Image Management International
 * 1100 Wayne Avenue
 * Silver Spring, MD 20910-5603
 * Tel: 301/587-8202
 * Fax: 301/587-2711
 * All Rights Reserved.
 * DMA (Document Management Alliance) working group.
 */

/* quexp2.cpp

This example is for the following query:

Find all priority 3 documents of any document class that
are filed in "blue" folders, and return the name of the
document and who filed it in the folder.

SELECT D.name, R.filer
FROM   F INNER JOIN R ON F.This = R.Tail
       INNER JOIN D (IncludeSubclasses=TRUE) ON R.Head = D.This
WHERE  F.color = "blue" AND D.priority = 3
ORDER BY <null>

0: F ("Folder") is a subclass of dmaClass_Container
1: R ("Ref. Cont. Relationship") is a subclass of
   dmaClass_ReferentialContainmentRelationship
2: D ("Document") is a subclass of dmaClass_DocVersion
*/

#define DMA_INIT_CD
#include <query.h>

/* Declare DmaId variables for the searchable class ID's.
*/
#define DVal { 0x12345678, 0x1234, 0x5678, \
              { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 } }
DMA_EXTERN_C DmaId D = DVal;

#define FVal { 0x12345678, 0x1234, 0x5678, \
              { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x09 } }
```

```
DMA_EXTERN_C DmaId F = FVal;

#define RVal { 0x12345678, 0x1234, 0x5678, \
              { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x09 } }
DMA_EXTERN_C DmaId R = RVal;

/* Declare DmaId variables for the property ID's.
*/
#define priorityVal { 0x12345678, 0x1234, 0x5678, \
                     { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x0A } }
DMA_EXTERN_C DmaId priority = priorityVal;

#define colorVal { 0x12345678, 0x1234, 0x5678, \
                  { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x0B } }
DMA_EXTERN_C DmaId color = colorVal;

#define filerVal { 0x12345678, 0x1234, 0x5678, \
                  { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x0C } }
DMA_EXTERN_C DmaId filer = filerVal;

#define nameVal { 0x12345678, 0x1234, 0x5678, \
                  { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x0A } }
DMA_EXTERN_C DmaId name = nameVal;

/* Declare the variables needed to perform the query.
*/

JoinExprElt          From2 [] =
{
    /* F[0] */
    { 0, &F, DMA_FALSE, NULL, NULL, 0, NULL, 0, NULL },

    /* INNER JOIN R[1] ON [0].This = [1].Tail */
    { 1, &R, DMA_FALSE, &dmaJoinOperator_Inner,
      &dmaQueryOperator_EqualObject,
      0, &dmaProp_This, 1, &dmaProp_Tail
    },

    /* INNER JOIN D[2] (IncludeSubclasses = TRUE)
       ON [1].Head = [2].This
    */
    { 2, &D, DMA_TRUE, &dmaJoinOperator_Inner,
      &dmaQueryOperator_EqualObject,
      1, &dmaProp_Head, 2, &dmaProp_This
    }
}
```

```
    },

    { -1, NULL, DMA_FALSE, NULL, NULL, 0, NULL, 0, NULL }
};

Property                Select2 [] =
{
    { 2, &name },
    { 1, &filer },
    { -1, NULL }
};

Operator                N12 = { &dmaQueryOperator_And };
Operator                N22 = { &dmaQueryOperator_EqualString };
Property                N32 = { 0, &color };
StringConst             N42 = { L"blue" };
Operator                N52 = { &dmaQueryOperator_EqualInteger32 };
Property                N62 = { 1, &filer };
IntegerConst            N72 = { 3 };

/* F.color = "blue" AND D.priority = 3
*/
QueryNode                QueryExpr2 [] =
{
    { OperatorNodeType, 0, &N12 },
    { OperatorNodeType, 1, &N22 },
    { PropertyNodeType, 2, &N32 },
    { IntegerConstNodeType, 2, &N42 },
    { OperatorNodeType, 1, &N52 },
    { PropertyNodeType, 2, &N62 },
    { StringConstNodeType, 2, &N72 },
    { StopperNodeType, -1, NULL }
};

QueryRoot                QR2 = { &From2[0], &Select2[0],
                                &QueryExpr2[0], NULL, DMA_FALSE };

DmaRC
QueryExample2(void)
{
    DmaRC                rc;
    IdmaScope *          pScope = NULL;
```

```
IdmaProperties *      pErrNode = NULL;
IdmaResultSet *      pResultSet = NULL;
IdmaProperties *      pResultRow = NULL;
DmaIndex32           SelectionsOffset;

/* TBD: Initializataion to get system object, doc space object,
   and target scope object = pScope.
*/

rc = StartQuery(&QR2, pScope, &pResultSet);
CHK(rc);

while (1)
{
    rc = pResultSet->GetNextResultRow(IID_IdmaProperties,
                                     (pDmapv)&pResultRow);

    if (rc != DMARC_OK)
    {
        break;
    }

    rc = pResultRow->GetPropValInteger32ById(
                                     &dmaProp_SelectListOffset,
                                     &SelectionsOffset);

    CHK(rc);

    /* TBD: Not shown: Access the values of the properties of
       the query result row referred to by pResultRow
       corresponding to the properties selected in the Selections
       list, e.g.,

           DmaIndex32    x;
           <datatype>    y;
           rc = pResultRow->GetPropVal<datatype>ByIndex(
                                   SelectionsOffset + x,
                                   &y);

       to store the value for selection list element x into y.
    */

    RELEASE(pResultRow); /* Release the Query Result row */
}

theend:
```

```
        return(DMARC_OK);  
    }  
}
```

5.4.8 QUERY3.CPP Example File

```
/*  
 * Copyright 1995, 1996, 1997 by the  
 * Association for Information and Image Management International  
 *      1100 Wayne Avenue  
 *      Silver Spring, MD 20910-5603  
 *      Tel: 301/587-8202  
 *      Fax: 301/587-2711  
 * All Rights Reserved.  
 * DMA (Document Management Alliance) working group.  
 */  
  
/* quexp3.cpp  
  
   This example enumerates the direct descendant objects directly  
   contained by a direct container object pX, as well as the  
   direct container objects that contain pX.  
*/  
  
#define DMA_INIT_CD  
#include <query.h>  
  
/* This procedure enumerates the immediate child objects that  
   are "directly" contained by the container, pParent.  
*/  
  
DmaRC  
EnumChildren(  
    IdmaProperties *      pParent)  
{  
    DmaRC rc;  
    IdmaEnumOfObject *    pChildren = NULL;  
    IdmaProperties *      pChild = NULL;  
    IdmaProperties *      pChildReln = NULL;  
    DmaIndex32 n;  
    IUnknown *          a [1];  
  
    a[0] = NULL;  
  
    rc = pParent->GetPropValObjectById(&dmaProp_Children,  
                                       IID_IdmaEnumOfObject,
```

```
(pDmapv)&pChildren);

CHK(rc);

while (1)
{
    /* Get the next child relationship object.
       The array "a" and scalar N are artifacts of the signature
       of GetNextObject().
    */
    rc = pChildren->GetNextObject(1, a, &n);
    if (rc != DMARC_OK)
    {
        break;
    }

    /* Since GetNextNext() didn't allow us to specify the interface
       we wanted, call QueryInterface() to get it.
    */
    rc = a[0]->QueryInterface(IID_IdmaProperties,
                              (pDmapv)&pChildReln);
    RELEASE(a[0]);
    CHK(rc);

    /* Get the actual child object pointed to by the
       relationship object.
    */
    rc = pChildReln->GetPropValObjectById(&dmaProp_Head,
                                          IID_IdmaProperties,
                                          (pDmapv)&pChild);
    RELEASE(pChildReln);
    CHK(rc);

    /* TBD: Process the child object pointed to by pChild */

    RELEASE(pChild);
}

rc = DMARC_OK;

theend:
    CHK_RELEASE(pChildren);
    CHK_RELEASE(pChild);
    return rc;
}
```

```
/* This procedure enumerates the container objects that immediately
   "referentially" contain the containee, pContainee.
*/
DmaRC
EnumContainers(
    IdmaProperties *          pContainee)
{
    DmaRC                    rc;
    IdmaEnumOfObject *       pContainers = NULL;
    IdmaProperties *         pContainer = NULL;
    IdmaProperties *         pContainerReIn = NULL;
    DmaIndex32               n;
    IUnknown *              a [1];

    a[0] = NULL;

    rc = pContainee->GetPropValObjectById(&dmaProp_Containers,
                                           IID_IdmaEnumOfObject,
                                           (pDmapv)&pContainers);

    CHK(rc);

    while (1)
    {
        /* Get the next container relationship object.
           The array "a" and scalar N are artifacts of the signature
           of GetNext().
        */
        rc = pContainers->GetNextObject(1, a, &n);
        if (rc != DMARC_OK)
        {
            break;
        }

        /* Since GetNext() didn't allow us to specify the interface
           we wanted, call QueryInterface() to get it.
        */
        rc = a[0]->QueryInterface(IID_IdmaProperties,
                                   (pDmapv)&pContainerReIn);

        RELEASE(a[0]);
        CHK(rc);

        /* Get the containing object pointed to by the
```



```
        relationship object.
    */
    rc = pContainerReIn->GetPropValObjectById(&dmaProp_Tail,
                                              IID_IdmaProperties,
                                              (pDmapv)&pContainer);

    RELEASE(pContainerReIn);
    CHK(rc);

    /* TBD: Process the container object pointed to by pContainer */

    RELEASE(pContainer);
}

rc = DMARC_OK;

theend:
    CHK_RELEASE(pContainers);
    CHK_RELEASE(pContainer);
    return rc;
}

void main(void)
{
    /* TBD: Declare variables pX, etc.
    Get system manager object.
    Get system object.
    Get doc space object.
    Get contained direct container object, pX.

    Call EnumChildren(pX).
    Call EnumContainers(pX).
    pX->Release();
    */
}
```

5.5 Containment Examples

- COSAMPLE1.CPP: demonstrates enumerating InitialContainer objects of a DocSpace.
- COSAMPLE2.CPP: demonstrates adding a DocVersion object to a container object.

5.5.1 COSAMPLE1.CPP Example File

```
/*
 * Copyright 1995,1996, 1997 by the
 * Association for Information and Image Management Int'l
 *
 *      1100 Wayne Avenue
 *      Silver Spring, MD 20910-5603
 *      Tel: 301/587-8202
 *      Fax: 301/587-2711
 *
 * All Rights Reserved.
 *
 *      DMA (Document Management Alliance) working group.
 *
 *
 * Containment Model code example:
 *
 *      Enumerate InitialContainer objects of a DocSpace in DMA 1.0
 *
 */

#ifndef NULL
#   define NULL    0
#endif

#include <stdio.h>
#include <dmatypes.h>
#include <dmacom.h>
#include <dmarc.h>
#include <dmaiface.h>
#include <dmaenums.h>
#include <dmaidvar.h>

//
// Assumes that connections to system and doc space have already been made
// (see "System and Document Space Connection Example")
//

////////////////////////////////////
////////////////////////////////////
```

```
void EnumInitialContainers(LPIdmaDocSpace pIDocSpace)
{
    DmaRC          rc;
    LPIdmaProperties pIPropsDocSpace = NULL;

    // Get the docSpace's properties interface
    rc = pIDocSpace->QueryInterface( IID_IdmaProperties,
                                     (pDmapv)&pIPropsDocSpace );

    if ( rc == DMARC_OK )
    {
        LPIdmaEnumOfObject pIEnumRootContainers = NULL;

        //
        // Get the list of initial containers in the docSpace using
        // the predefined DMA property Id -- dmaProp_InitialContainers
        //

        rc = pIPropsDocSpace->GetPropValObjectById(
                                     &dmaProp_InitialContainers,
                                     IID_IdmaEnumOfObject,
                                     (pDmapv)&pIEnumRootContainers );

        if ( (rc == DMARC_OK) && (pIEnumRootContainers != NULL) )
        {
            DmaIndex32  nNumReturn;
            IUnknown     *pIRootContainer;

            // Now enumerate each of these root containers one at a time

            rc = pIEnumRootContainers->GetNextObject( 1, // get one item
                                                       &pIRootContainer,
                                                       &nNumReturn );

            while ( (rc == DMARC_OK) && (nNumReturn == 1) )
            {
                //
                // ### Process this initial container here! ###
                //     For example, the container may be
                //     enumerated to list the objects contained in it.
                //

                // Release the initial container interface
            }
        }
    }
}
```

```
pIRootContainer->Release();

// Get the next initial container in the enumerator
rc = pIEnumRootContainers->GetNextObject( 1, // get one item
                                           &pIRootContainer,
                                           &nNumReturn );
}

// Release the enumerator
pIEnumRootContainers->Release();
}

// Release the docSpace properties interface
pIPropsDocSpace->Release();
}
}

////////////////////////////////////
////////////////////////////////////
```

5.5.2 COSAMPLE2.CPP Example File

```
/*
 * Copyright 1995,1996, 1997 by the
 * Association for Information and Image Management Int'l
 *
 *      1100 Wayne Avenue
 *      Silver Spring, MD 20910-5603
 *      Tel: 301/587-8202
 *      Fax: 301/587-2711
 *
 * All Rights Reserved.
 *
 *      DMA (Document Management Alliance) working group.
 *
 *
 * Containment Model code example:
 *
 *      Adding a DocVersion object into a Container object
 *      using direct containment relationship DMA 1.0
 *
 */

#ifndef NULL
#  define NULL    0
#endif

#include <stdio.h>
#include <dmatypes.h>
```

```

#include <dmacom.h>
#include <dmarc.h>
#include <dmaiface.h>
#include <dmaenums.h>
#include <dmaidvar.h>
#include <dmacaps.h>

//
// Code example: Adding a DocVersion object into a Container object using
// direct containment relationship.
//
// Also, demonstrates the use of the docspace capabilities to determine
// support for direct containment.
//
// Assumes that connections to system and doc space have already been made
// (see "System and Document Space Connection Example") and that the doc id
// of the target document has been found (by search or navigation)
//

////////////////////////////////////
////////////////////////////////////

DmaRC
ReportError(DmaRC rc, char *msg)
{
    // Do stuff here to display error msg
    return DMARC_OK;
}

////////////////////////////////////
////////////////////////////////////

DmaRC
Abort(DmaRC rc, char *msg)
{
    ReportError(rc, msg);
    return (rc);
}

////////////////////////////////////
////////////////////////////////////

DmaRC
Message(char *pszMessage)

```

```

{
    // information message display code here
    return DMARC_OK;
}

////////////////////////////////////
////////////////////////////////////

DmaRC
CheckContainmentCapability (LPIdmaDocSpace    pIDocSpace)
{
    DmaRC                rc;
    LPIdmaProperties      pIDocSpaceProps = NULL;
    DmaInteger32          nCapabilityValue = DMA_TRUE; // default is set
    LPIdmaListOfInteger32 pICapabilitiesList;

    // Get the properties interface from the docspace
    rc = pIDocSpace->QueryInterface ( IID_IdmaProperties,
                                      (pDmapv)&pIDocSpaceProps );
    if ( rc != DMARC_OK )
    {
        Abort (rc, "Cannot get properties interface on DocSpace");
        return DMARC_ABORT;
    }

    // Get the DocSpace Capabilities property
    // Note: we must be authenticated to the docspace
    // to obtain any object valued properties.
    rc = pIDocSpaceProps->GetPropValObjectById (
                                                &dmaProp_DocSpaceCapabilities,
                                                IID_IdmaListOfInteger32,
                                                (pDmapv)&pICapabilitiesList );

    if ( (rc != DMARC_OK) || (pICapabilitiesList == NULL) )
    {
        Abort (rc, "Cannot get the DocSpace's Capabilities property");
        pIDocSpaceProps->Release();
        return DMARC_ABORT;
    }
    else
    {
        // Get the capability value for direct containment
        // Note: general docspace capabilities such as being a writable
        // docspace, can all be determined via the various capability

```

```
// list values.
rc = pICapabilitiesList->GetInteger32 ( DMAC_CONTAIN_DIR,
                                         &nCapabilityValue );
if ( ( rc == DMARC_OK ) && ( nCapabilityValue == DMA_TRUE ) )
{
    // direct containment is supported
    pIDocSpaceProps->Release();
    pICapabilitiesList->Release();
    return DMARC_OK;
}
else
{
    pIDocSpaceProps->Release();
    pICapabilitiesList->Release();
    return DMARC_FAILED;
}
}
}

////////////////////////////////////
////////////////////////////////////

DmaRC
InsertDocVersionIntoContainer(
    pDmaString      DocumentId,
    pDmaString      pidContainer,
    LPIdmaSystem    pISystem,
    LPIdmaDocSpace  pIDocSpace )
{
    DmaRC          rc;
    LPIdmaProperties pIDocVer          = NULL;
    LPIdmaProperties pIContainer       = NULL;
    LPIdmaObjectFactory pIObjectFactory = NULL;
    LPIdmaEditProperties pIDirectRelationship = NULL;
    LPIdmaListOfObject pIChildrenList  = NULL;
    LPIdmaConnection pIConnection     = NULL;
    LPIdmaObject pIFirstRelationship  = NULL;
    LPIdmaBatch pIBatch               = NULL;
    LPIdmaRelationshipOrdering pIDirectRelationshipOrdering = NULL;

    rc = CheckContainmentCapability ( pIDocSpace );
    if ( rc != DMARC_OK )
    {

```

```
        Abort (rc, "Document space does not support direct containment");
        // Note: must release all interface pointers obtained prior to
        // any of the abnormal returns as below.
        return DMARC_ABORT;
    }

    rc = pIDocSpace->ConnectObject( DocumentId,
                                    IID_IdmaProperties,
                                    (pDmapv)&pIDocVer );

    if ( rc != DMARC_OK )
    {
        Abort (rc, "Failed to connect to doc ver");
        return DMARC_ABORT;
    }

    // NOTE: for brevity, further error checking is omitted. All DMA calls
    // should be followed by code similar to the above

    rc = pIDocSpace->ConnectObject ( pidContainer,
                                    IID_IdmaProperties,
                                    (pDmapv)&pIContainer );

    // Next, Get the DocSpace IObjectFactory interface,
    // then create a new direct containment relationship object.

    rc = pIDocSpace->QueryInterface( IID_IdmaObjectFactory,
                                    (pDmapv)&pIObjectFactory );
    if ( rc != DMARC_OK )
    {
        // This could have been determined by checking the
        // the appropriate capability value.
        Abort(rc, "DocSpace does not allow object creation");
        return DMARC_ABORT;
    }
    rc = pIObjectFactory->CreateObject(
                                    &dmaClass_DirectContainmentRelationship,
                                    IID_IdmaEditProperties,
                                    (pDmapv)&pIDirectRelationship );

    // Check to see if this failed because the docspace
    // doesn't support direct containment

    if ( rc != DMARC_OK )
    {
```



```
        Abort (rc, "Failed to get a direct containment relationship");
        return DMARC_ABORT;
    }

    // Set the tail and head of the relationship object to point to the
    // container and docVersion, respectively
    rc = pIDirectRelationship->PutPropValObjectById( &dmaProp_Head,
                                                    pIDocVer );

    rc = pIDirectRelationship->PutPropValObjectById( &dmaProp_Tail,
                                                    pIContainer );

    // Get an interface to insert the relationship
    rc = pIDirectRelationship->QueryInterface( IID_IdmaRelationshipOrdering,
                                              (pDmapv)&pIDirectRelationshipOrdering );

    // release our property interface
    pIDirectRelationship->Release();

    // For this example, we want to insert this as the first relationship
    // in the containing folder. But we don't care where the relationship is
    // inserted on the document end (Because this is a direct containment
    // relationship, this will be the only direct relationship connecting
    // this document with this folder)

    // Get the first relationship in the folder's dmaProp_Children list.
    rc = pIContainer->GetPropValObjectById( &dmaProp_Children,
                                           IID_IdmaListOfObject,
                                           (pDmapv) &pIChildrenList );

    rc = pIChildrenList->GetObject( 0,
                                   IID_IdmaObject,
                                   (pDmapv)&pIFirstRelationship );

    rc = pIDirectRelationshipOrdering->SetOrdering( pIFirstRelationship,
                                                    DMA_POS_FIRST,
                                                    NULL,
                                                    DMA_POS_ANYWHERE );

    //
    // In order to make this containment relationship as persistent
    // Invoke the IdmaBatch::ExecuteChanges() on the DocSpace to save this
    // object into the repository
```

```
//
rc = pIDocSpace->QueryInterface( IID_IdmaBatch,
                                (pDmapv)pIBatch );

if ( DMARC_OK == rc )
{
    // Create a List object to hold the objects need to save change back
    // to the repository
    LPIdmaEditListOfObject pIChangeList;
    rc = pIObjectFactory->CreateObject(
                                &dmaClass_ListOfObject,
                                IID_IdmaEditListOfObject,
                                (pDmapv)&pIChangeList );

    if ( DMARC_OK == rc )
    {
        // Insert the new relationship object to the change list
        rc = pIChangeList->InsertObject( 1,
                                pIDirectRelationshipOrdering );

        if ( DMARC_OK == rc )
        {
            //
            // Execute Changes to save to the repository
            //
            DmaIndex32 ulIndex;
            rc = pIBatch->ExecuteChanges( NULL, // no callback procedure
                                pIChangeList,
                                &ulIndex,
                                DMA_MODIFY_UNPROTECTED,
                                DMA_FALSE );    // Don't refresh objects

            if ( DMARC_OK == rc )
            {
                //
                // ### the docVersion is inserted into the container ###
                //
            }
            else
            {
                // Do your error handling here
                ReportError(rc, "Cannot save the relationship" );
            }
        }
    }
    else
    {
        // Do your error handling here
        ReportError(rc, "Cannot insert the relationship object");
    }
}
```

```
    }

    // Release the List interface
    pIChangeList->Release();
}

// Release the IdmaBatch interface
pIBatch->Release();
}

// Release docSpace's object factory interface
pIObjectFactory->Release();

// release interface pointers
pIChildrenList->Release();
pIDirectRelationshipOrdering->Release();
pIDirectRelationship->Release();
pIContainer->Release();
pIDocVer->Release();
pIConnection->Release();

return DMARC_OK;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

5.6 Versioning Example

```
/*
 * Copyright 1995,1996,1997 by the Association for Information and Image Man-
 * agement Int'l
 *
 *                               1100 Wayne Avenue
 *                               Silver Spring, MD 20910-5603
 *                               Tel: 301/587-8202
 *                               Fax: 301/587-2711
 *
 * All Rights Reserved.
 * DMA (Document Management Alliance) working group.
 *
 * Code example: Versioning Example
 *
 */

#include <dmatypes.h>
#include <dmacom.h>
#include <dmarc.h>
#include <dmaenums.h>
#include <dmaiface.h>
#include <dmaenums.h>
#include <dmaids.h>
#include <dmaidvar.h>
#include <dmacfunc.h>

#ifndef NULL
#define NULL    0
#endif

// This example creates a new versioned entity (Configuration History
// and primary Version Series), then makes a Reservation against the
// version series and checks in a new DocVersion.
//
// The following function is assumed to exist outside this module
// for initialising (but not saving) a new DocVersion object. See
// the content samples to see how this might be done.
extern DmaRC PrepareNewDocVer ( IdmaObjectFactory *pIFactory,
                               DMA_REFIID riidDovVer,
                               pDmapv ppIDocVer );

// Forward declarations
DmaRC DoExecuteChange ( IUnknown *punkToSave );

DmaRC VersionExample (
```

```
IdmaObjectFactory *pIFactory )
{
    // Step 1: Create a Configuration History object
    IdmaEditProperties *pedpropCfg;
    DmaRC rc = pIFactory->CreateObject ( &dmaClass_ConfigurationHistory,
                                         IID_IdmaEditProperties,
                                         (pDmapv)&pedpropCfg );

    if ( rc != DMARC_OK )
        return rc;

    // All remaining error handling omitted

    // Set any extended properties of Config History
    // *** None in this example ***

    // Step 2: Create a Version Series object
    IdmaEditProperties *pedpropVerSer;
    rc = pIFactory->CreateObject ( &dmaClass_VersionSeries,
                                   IID_IdmaEditProperties,
                                   (pDmapv)&pedpropVerSer );

    // Step 3: associate Version Series with Config History as primary
    rc = pedpropVerSer->PutPropValObjectById ( &dmaProp_ConfigurationHistory,
                                              pedpropCfg );
    rc = pedpropVerSer->PutPropValBooleanById ( &dmaProp_IsPrimarySeries,
                                              DMA_TRUE );

    // Step 4: save both objects. Note that the order is important
    rc = DoExecuteChange ( pedpropCfg );
    rc = DoExecuteChange ( pedpropVerSer );

    // The Config History object is not required in what follows.
    pedpropCfg->Release();

    // Get the Version Series interface
    IdmaVersionSeries *pIVerSer;
    rc = pedpropVerSer->QueryInterface ( IID_IdmaVersionSeries,
                                         (pDmapv)&pIVerSer );

    // The EditProperties interface is no longer required
    pedpropVerSer->Release();

    // Step 5: Create a Reservation object
    IdmaEditProperties *pedpropRes;
    rc = pIFactory->CreateObject ( &dmaClass_Reservation,
```

```

        IID_IdmaEditProperties,
        (pDmapv)&pedpropRes );

// Set any extended properties of Reservation object
// *** None in this example ***

// Step 6: Make the reservation against the version series
pIVerSer->SetReserveNext ( pedpropRes );
pedpropRes->Release();      // no longer required
DoExecuteChange ( pIVerSer );

// Step 7: Prepare new Version Description
IdmaEditProperties *pedpropVerDesc;
rc = pIFactory->CreateObject ( &dmaClass_VersionDescription,
                               IID_IdmaEditProperties,
                               (pDmapv)&pedpropVerDesc );

// Set any extended properties of Version Description object
// *** None in this example ***

// Step 8: Prepare new DocVersion
IdmaVersionable *pverDocVer;
rc = PrepareNewDocVer ( pIFactory, IID_IdmaVersionable,
                       (pDmapv)&pverDocVer );

// Step 9: Make the checkin
rc = pverDocVer->SetCheckIn ( pIVerSer, pedpropVerDesc );
pIVerSer->Release();
pedpropVerDesc->Release();
rc = DoExecuteChange ( pverDocVer );
pverDocVer->Release();

return rc;
}

DmaRC DoExecuteChange (
    IUnknown *punkSave )
{
    IdmaConnection *pconnSave;
    DmaRC rc = punkSave->QueryInterface ( IID_IdmaConnection,
                                          (pDmapv)&pconnSave );

    if ( rc == DMARC_OK )
    {
        rc = pconnSave->ExecuteChange ( NULL,

```

```
        DMA_MODIFY_UNPROTECTED,  
        DMA_TRUE );    // refresh  
  
    pconnSave->Release();  
}  
  
return rc;  
}
```

5.7 DMA Glossary

-A-

Access	The ability to view document-based information after passing existing authorization and authentication tests.
Addressability	The DMA concept of being able to locate and retrieve independently persistent objects in a DMA document space using their OIID's
Administration	The ability to reconfigure document management system components, monitor system operations, and improve system performance.
AIIM	Association for Information and Image Management International.
American National Standards Institute	(ANSI) A standards-setting, non-government organization that develops and publishes standards for "voluntary" use in the United States. Standards set by national organizations are accepted by vendors in that country. See http://www.ansi.org/ for additional information.
Annotation	The ability to capture user-generated information relating to a document.
ANSI	American National Standards Institute.
API	Application Program Interface.
Application Interface	DMA Application Interface.
Application Program Interface	(API) A functional interface supplied by the operating system (OS) or by a program that allows an application program to use specific data or functions of the OS or program.

Architecture	The specifications that detail the system design and components used to implement applications, providing a blueprint to assist developers during design and construction. The specifications that detail all of the technologies utilized in the delivery of solutions.
Archive (1)	A feature of Document Management systems, in which infrequently accessed documents are, moved to off-line or near-line storage areas.
Archive (2)	A copy of data on disks, CD-ROM, magnetic tapes, etc., for long term storage and later possible access.
Association for Information and Image Management International	<p>Accredited by ANSI as a standards-development organization, AIIM represents the US in the International Organization for Standardization, and is an umbrella organization for industry coalitions of vendors and end users working to develop worldwide interoperability standards for document management technologies.</p> <p>(See http://www.aiim.org/ for additional information.)</p>
Asynchronous Transaction	<p>(1) The ability of a client to make a request to a server to perform an operation where control is returned to the client while the server is performing the operation. This allows the client to do other processing while it's waiting for the server operation to complete.</p> <p>(2) An operation in which program control is returned to the caller prior to the completion of the requested operation. The value returned from a call to an asynchronous transaction usually indicates the result of attempting to start the operation. The caller must establish some additional mechanism to determine the completion status of the operation. Common mechanisms for receiving notification of completion are execution of callback functions, and polling by the calling application.</p>
Attributes	The descriptive information about a document or another object. Depending on the Document Management system, it may or may not include document content.
Authentication	Refers to determining the identity of the user attempting the access.

Authoring	The process of creating content that may be managed by a Document Management system.
-----------	--

Authorization	Refers to determining the set of privileges available to the user.
---------------	--

- B -

Backup	A process, either scheduled or ad hoc, to copy data and files to another storage subsystem, usually optical or tape. Either all files or recently modified files are marked for backup.
--------	---

Batching / Batch	The mechanism provided in DMA that allows changes to a set of objects to be made persistent in one atomic operation. Note however that the DMA batch mechanism does not have provide all the attributes required of a transaction mechanism.
------------------	--

Binary Large Object	(BLOB) A term used to refer to any random large block of data that is stored in a database, such as a video clips, sound files or document content data. Typically, database management systems provide methods to create, delete or replace these objects, without the ability to interpret the contents of these objects.
---------------------	---

Black Forest Group	A group founded in 1992, consisting of representatives from industry and academia. The groups goal is to provide a forum for the interchange of ideas and to provide direction for and influence the development of products and methods to solve the business problems of member companies.
--------------------	--

BLOB	Binary Large Object
------	---------------------

Branched Versioning	A scheme of maintaining multiple related ordered lists of versions of an entity. Each list is usually ordered by the time of insertion into that list of versions. Each version represents the state of the entity at some point in time. In addition, the relationships among the multiple lists are also recorded.
---------------------	--

- C -

Capabilities	DMA Capabilities.
Capture	The acquisition of documents through conversion of hardcopy formats, such as paper, microfilm, and microfiche, into an electronic format.
CD-ROM	Compact Disc Read Only Memory. An optical technology for storing data. CD-ROMs currently hold more than 600 megabytes of data. Their low cost enables mass distribution of data.
Character Set	A collection of elements used to organize, control, or represent symbolic information.
Character Set Encoding	A character set in which each character is assigned a numeric code value. Frequently abbreviated as character set when the context is sufficient to determine what is intended.
Check-in / Check-out	<p>A Document Management system feature that coordinates document updates among multiple users. Check-in and check-out functions can be defined to support a wide range of versioning and collaborative authoring schemes.</p> <p>In DMA, Check-in is the action of adding a new version to a Version Series. The client must have a Reservation against the specified version series.</p> <p>Check-out is the action of locking the right to create the next version in a version series and requesting a copy of the current version's material to be made.</p>
Child	A DMA object that is directly contained within a container as opposed to a referentially contained object that is called a Containee object in DMA.

Class Factory	<p>An object that manufactures new objects of a particular class ID by exposing the well defined COM IClassFactory interface. A class factory performs a type of bootstrapping for clients wanting access to a COM component by creating the “root” object to a component and providing an interface to it. Thereafter the client can navigate the object hierarchy of the COM component using the standard interfaces.</p> <p>See Object Factory and Scope Factory for DMA object creation capabilities.</p>
Code Point	<p>Numeric value assigned to a character in a particular character set encoding.</p>
Code Set	<p>Character Set Encoding.</p>
COLD	<p>Computer Output to Laser Disk.</p>
Collaborative Authoring	<p>The process of allowing a group of people to author a document collection together, even though they may be located in different places or working at different times.</p>
COM	<p>Component Object Model.</p>
Component Object Model	<p>(COM) Microsoft's Component Object Model specification, which describes objects and interfaces in a language and location independent manner. COM facilitates run-time discovery of objects and interfaces and thus allows applications to be built from binary software components. It is the foundation for Microsoft's OLE technology.</p>
Compound Document	<p>A compound document is a composite electronic document that is made up of a number of components. Typically, these components are in a variety of formats from many different sources.</p>
Computer Output to Laser Disk	<p>(COLD) A means of converting report data on legacy mainframe systems to text documents on a client-server system.</p>
Conceptual Entity	<p>Conceptual Versioned Entity.</p>

Conceptual Versioned Entity	The ensemble of an entity (for instance a set of Document Versions) and their supporting objects which represent the history of the entity, including branches and merges, over time.
Configuration History	A DMA object used in support of DMA Versioning. A configuration history is used as the root object by which all of the components of a conceptual entity are reachable through navigation. Through sub-classing, it may also contain properties that are applicable to the entire conceptual entity.
Containable	The generic class of DMA objects that are allowed to be contained within containers.
Containee	A DMA object that is referentially contained within a container as opposed to a directly contained object, which is, called a child object in DMA.
Container	<p>Generally, an object that facilitates object aggregation and navigation of a collection of objects that are related to one another through a "contains" relationship.</p> <p>In DMA, a Container is an object that can contain other objects either directly or by reference.</p>
Containment	<p>The process of containing one or more objects inside another (Container) object. Generally in document management systems grouping documents into "folders", "cabinets" etc. is referred to as containment and is a means of organizing documents that are related in some way.</p> <p>See Direct Containment. and Referential Containment.</p>
Containment Model	The mechanism provided in DMA to represent containment and provide associated capabilities to a DMA application through a well-defined set of objects.
Containment Relationship	The DMA object that facilitates capturing of edge data in a containment relationship. Various properties about the relationship between the Container and Containee or other relevant information can be held by this object. E.g. "Inserted By" (who inserted the containee into the container), "Insertion Date" (when it was inserted). The two sub-classes, Direct Containment Relationship and Referential Containment Relationship, have been defined to model the specific types of containment supported in DMA.

Content	Generally, the data portion, as opposed to the attributes, of a document. Some Document Management systems regard content as just another attribute. In DMA, the content of a specific document is defined by a collection of objects that are associated to the document and can be accessed as defined by the Content Model.
Content Element	<p>The base class of DMA objects that are used to access document content. A document version's, Rendition consists of one or more content elements.</p> <p>Two subclasses, Content Transfer and Content Reference, of Content Element have been defined to facilitate the different forms of content capture and access.</p>
Content Material	Content.
Content Model	The mechanism provided in DMA to represent the content of documents within a DMA system. The model defines the characteristics and behavior of a set of objects that a DMA application can use to store and retrieve content data of documents in a format neutral manner.
Content Reference	A DMA object that represents an elementary component of document content that exists outside the control of a document space, to which a reference is maintained. This is a subclass of DMA content element and is used by a client to store a reference to the content data. The reference is a resource name in URL format and it is the client's responsibility to gain access to it.
Content Transfer	A DMA object that represents an elementary component of document content that is directly captured and managed by a document space. This is a subclass of DMA content element and is used by a client to, transfer content data to a document space and subsequently access it.

Conversion	To change the format of a document, or a component within a document. The act of conversion may be further classified into types of conversion – conversion between character sets, conversion between word processor formats, or conversion between different page description languages. Conversions that actually change the logical structure of a document are frequently referred to as document transformations. This distinction is used to indicate that changes in the logical structure of a document may result in the addition, deletion, or reordering of document components; e.g., adding required elements to create a parse-able instance of an SGML document.
Controlling Object	A DMA object that exposes and controls a metadata space. For example, the system object is the controlling object for the system metadata space while the scope object is the controlling object for the scope metadata space.
Creation	The function of adding attribute information and content, either in the form of an original document, or as one derived from an existing document.
Custom Properties	Extended Properties.

- D -

Database	A collection of data with a given structure for accepting, storing, and providing data on demand. Typically, databases provide a more robust environment for the storage of persistent data than that provided by OS file systems. Characteristics of databases include multi-user concurrency controls, journalling, replication, data dictionaries, user definable schema, strong data typing, and sophisticated query languages.
DDE	Dynamic Data Exchange.
Delivery / Use	The delivery of documents and associated information on demand within a distributed environment to users who participate in document-based business processes.

Dependently Persistent Object	See Persistent Object.
Digital Photo	Photographic images captured in a digital format, rather than on film.
Digital Signature	A mark, encrypted or unencrypted, used in the approval process for a document.
Digital Video	Video captured in a digital format, rather than on tape.
Distribution Media	Output media for storage and replication of documents.
Direct Containment	DMA model for representing containment where a container may contain multiple containees but a containee is contained in at most one container. This essentially models a 1:N relationship. The terms parent and child are used to refer to the containers and containees that participate in this form of containment.
DMA	Document Management Alliance.
DMA Application Interface	<p>The DMA Application Interface is the uniform programmatic interface that an application must use to access DMA Systems and DMA Service Elements from within any operating environment.</p> <p>All DMA programmatic interfaces are defined as a set of COM Objects and Interfaces with the exception of two standalone functions defined in the DMA integration model.</p> <p>DMA makes use of the following COM features and does not depend on any COM libraries: Globally Unique Identifiers, Object and Interface Mechanism, Life-cycle model with reference counting and IUnknown being the standard base interface for all DMA COM objects.</p>
DMA Capabilities	The definition of groups of functionality and associated set of elements of the DMA specification that DMA systems must implement in order to provide a specific capability.
DMA Certification	Document Management Systems and Applications that have been endorsed as satisfying the DMA conformance requirements.

DMA COM API	DMA Application Interface.
DMA COM Interface	DMA Application Interface.
DMA Compliance	The correspondence of DMA products to a well-defined set of conformance criteria. By definition, DMA product components that comply with the same conformance criteria are expected to be interoperable.
DMA Conformance	The definition of levels or groups of capabilities to help achieve interoperability between various DMA product components.
DMA Coordination Layer	DMA Middleware.
DMA Middleware	Middleware.
DMA Object	A COM object supporting at least the minimum required DMA interfaces.
DMA System	System.
DMA URL	<p>DMA URL's describe the identity of a DMA object and a mechanism to locate a DMA object in a hierarchical manner relative to a DMA System and DMA Document Space.</p> <p>DMA URL's are consistent with and follow the standard URL syntax.</p> <p>An OIID is a DMA URL.</p>
Document	A collection of information that pertains to a particular subject or related subjects.
Document Content Model	A specification that defines the characteristics of a single or multi component document. E.g. OpenDoc, SGML and the DMA Content Model are examples.
Document Interchange Format	The rules for representing documents for the purpose of interchange.

Document Management	The total set of processes, people, standards, tools, and systems to make effective use of documents.
Document Management Alliance	(DMA) The Document Management Alliance is an AIIM task force, consisting of user companies and document management vendors, dedicated to developing a specification for the universal interoperability of all document management applications and repositories. DMA was formed from the merger of two previous standards groups (Shamrock and DEN).
Document Version	The DMA representation of a document, which consists of the content, state and descriptive information about the document presented in the form of Properties.
Document Property	The DMA representation of each attribute of a document. The Properties and Property Values are name/value pairs that describe each attribute in a Document Version. Document Versions have well defined properties and can have custom properties due to the extensible nature of DMA.
Document Retrieval	The ability to search for, select and use a document from a document management repository.
Document Space	The DMA representation of a document collection or repository. A Document Space generally determines technology, capabilities and defines the policies for the management of documents within it. Typically, DMA application users will interact with one or more Document Spaces by searching, retrieving and storing documents.
Document Space Scope	See Scope.
Document System	The DMA representation of a group of Document Spaces available to a DMA application at given point of access. The Document System provides a single point of contact for applications to determine which Document Spaces are available and their capabilities.
Dynamic Data Exchange	(DDE) Dynamic Data Exchange. A single-node, inter-process messaging protocol developed by Microsoft for use in the Microsoft Windows family of operating systems.

- E -

Edge Data	Properties that are specific to a relationship between two objects. These additional properties are not part of the actual objects themselves and typically are captured and represented by a separate object. In DMA, the Relationship objects capture this edge data.
EDI	Electronic Data Interchange.
EDM (1)	Engineering Document Management.
EDM (2)	Enterprise Document Management.
EDM (3)	Electronic Document Management.
Electronic Data Interchange	(EDI) The exchange of data and documents between different users according to standardized (ANSI X.12, EDIFACT) rules.
Embedding	A means of incorporating data in a compound document, whereby the data and the association with its managing application are physically located within the compound document.
Engineering Document Management	(EDM 1) The management of engineering-related data and documents.
Enterprise	A corporate user base, typically operating within a LAN/WAN environment and encompassing an entire organization, therefore containing multiple diverse groups that may have different and potentially incompatible computer systems.
Enterprise Document Management	(EDM 2) The management of document-based information across an enterprise.

Enterprise Document Management Services	The services, independent of any specific application domain, that support document management needs.
---	---

Extended Properties	(Custom Properties) Properties that exist in the metadata of DMA document spaces, but that are not defined in the DMA specification. A document space may provide extensions either, by adding properties to any of the DMA defined classes or by defining subclasses with additional properties.
---------------------	---

- F -

Fax	The use of a telephone system for the electronic transmission and receipt of hard copy images, utilizing CCITT Group 3 or 4 compression.
-----	--

Filter	A process that modifies stored data for display purposes. See Conversion.
--------	---

Forms Processing	A type of data processing facilitated by forms.
------------------	---

Full-text Retrieval	A document search method based on document text content.
---------------------	--

- G -

Globally Unique Identifier	(GUID) The term used in COM for Universally Unique Identifiers.
----------------------------	---

GroupWare	Software that allows people on the network to participate in a joint project.
-----------	---

GUID	Globally Unique Identifier.
------	-----------------------------

- H -

HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
Hypertext	A way of presenting information on-line with connections between one piece of information and another called hypertext links.
Hypertext Markup Language	(HTML) A markup language for defining the structure of documents published on the World Wide Web (www); it is an SGML application. Developed at CERN, HTML is continually evolving with extensions being proposed and supported by vendors such as Netscape and Microsoft under the direction of the W3C.
Hypertext Transfer Protocol	(HTTP) The most commonly used application-level protocol on the World Wide Web, ideally suited for search and retrieval and transferring Hypertext.

- I -

IANA	Internet Assigned Numbers Authority.
I18N	Internationalization (due to the 18 characters between the first and last character in the word).
ID Model	The means by which all objects in a DMA document space are uniquely identified. These unique identifiers are referred to as GUID's and it is requisite that every independently persistent object in a DMA document space must have a GUID.
IETF	Internet Engineering Task Force.

Internet Engineering Task Force	<p>(IETF) A task force sponsored by the Corporation for National Research Initiatives, which is organized into various working groups each focusing on defining standards for the Internet.</p> <p>See http://www.ietf.org/ for additional information.</p>
Image Management	<p>A system designed to handle the requirements of image-based documents.</p>
Immediate Subclass	<p>A proper subclass that has been directly derived from another class is referred to as an immediate subclass of the class from which it was derived from. In other words, the class from which it was derived from is its immediate superclass.</p>
Immediate Superclass	<p>A DMA class, which is used to directly derive the definition of another class, is referred to as the immediate superclass of the derived class. In other words, there are no intermediate classes in the single inheritance hierarchy between the derived class and its immediate superclass.</p> <p>An immediate superclass of a derived class is also one of its superclasses.</p>
Independently Persistent Object	<p>See Persistent Object.</p>
Indexing	<p>The process of associating attributes with a document for retrieval purposes. The process of creating data structures to speed the searching of attributes; e.g., "create an index on the title field".</p>
Industry Application Services	<p>Sets of user-defined services common to a given industry's core activities.</p>
Information Objects	<p>The building blocks of the content of a document; e.g., text, graphics, equations, vector data, data base extracts, photographs, audio, video, executable processes, and references to hard copy and other media.</p>

Inheritance	<p>The object oriented technique of defining a new class using one or more existing classes as a model. Generally new specialized classes are derived from existing more generic classes.</p> <p>DMA supports inheritance by allowing a new class to be defined using a single existing class. The new class must have the methods and properties of the class it is derived from, and is allowed to add new methods and properties.</p>
Integration Model	<p>The parts of the DMA architecture that define how service elements and systems are "integrated" into the architecture. This specifies the registration mechanism for service elements and how they are accessed by the clients of the interface.</p>
Internationalization	<p>The design of computer software for users around the world such that it is adaptable to the requirements of different native languages, local customs, and character-string encodings. Normally a goal of internationalization is to permit localization by users or administrators at a particular location.</p>
Internet Assigned Numbers Authority	<p>(IANA) The central coordinator for the assignment of unique parameter values for Internet protocols. These include internet addresses, domain names, protocol numbers, port numbers, and many others.</p> <p>See http://www.isi.edu/div7/iana/ for additional information.</p>

- J K L -

Library Services	<p>A set of features of Document Management systems, including document check-in/check-out, version control, and access control.</p>
Life Cycle / Final Distribution Management	<p>A Document Management feature, whereby a document's creation, review, publishing, archiving, and purging are managed.</p>
Linear Versioning	<p>A scheme of maintaining a single ordered list of versions of an entity. The list is usually ordered by the time of insertion into the list of versions. Each version represents the state of the entity at some interesting point in time.</p>

Linking A means of incorporating objects in a compound document, whereby a link reference is inserted into the document pointing to the actual data, which physically resides elsewhere in the document or in some other document.

Locale Specifies the native language, character set encoding scheme and other attributes specific to a particular country or location. Locale's are used to provide an application's capabilities in a localized form.

Localization The process of adapting the operation of computer software to a particular native language, local custom, or string encoding. Typically, this is done at the location the software is being used without the need to modify or recompile the source program.

- M -

MAPI Messaging API.

Merged Scope This provides one of the key features of DMA, that of coordinated distributed search. A merged scope object is constructed from a list of component scope objects from multiple document spaces. The merged scope object presents a unified view of the metadata of the component scope objects. This is then used to formulate and distribute queries to those document spaces and to retrieve merged result sets from the queries.

This capability is implemented by a Scope Factory service element.

Messaging API (MAPI) A middleware messaging standard, supported by Microsoft and typically used by mail products, that provides a client API and a service provider API in order to insulate application developers from the details of the underlying messaging system.

Metadata	<p>Generally refers to data about data.</p> <p>In DMA, metadata is used extensively to facilitate the runtime discovery of DMA objects and their properties. Every DMA object describes the class to which it belongs by means of an object containing a description of the class. The properties of a class are described by property description objects.</p>
Metadata Space	<p>A metadata space is a set of classes that form a single-rooted inheritance hierarchy. At runtime, a metadata space is exposed by what is termed a controlling object and is represented as a collection of fully connected class description objects of member classes in the metadata space.</p> <p>The DMA specification defines a number of different types of metadata spaces, which contain subsets of the classes defined in this specification and other subclasses of those. Some examples are, system metadata space, document space metadata space and scope metadata space.</p>
Middleware	<p>Software that shields a distributed application developer from the complexities of the hardware, operating system, and network semantics.</p> <p>In a DMA system, Middleware is an essential component, which manages the distribution of access between DMA applications and service elements. Middleware provides an implementation of the DMA system object and the means for registering and locating service elements at a point of presence.</p> <p>The DMA middleware is also referred to as the DMA coordination layer.</p>
MIME	<p>Multipurpose Internet Mail Extensions.</p>
MIME Content-Type	<p>Is the means of specifying the type and sub-type of content information including fully specifying the native representation of the data that constitute the content. Originally defined as part of MIME, its use is not limited to Internet mail.</p> <p>The HTTP protocol used on the WWW uses MIME content-types to specify the type of files delivered to web browser's, with the browser's using this information to launch appropriate handlers or applications.</p> <p>DMA also makes use of MIME content-types to specify the content stored in Renditions.</p> <p>Mime content-types are extensible and registration is handled by IANA.</p> <p>See predefined MIME content-types for additional information.</p>

Multipurpose Internet Mail Extensions (MIME) A specification that provides a way to interchange multimedia E-mail among many different computer systems that use Internet mail standards. MIME is being used in many other applications than E-mail with MIME content types being widely used on the WWW and in DMA.

See MIME RFC 1521 and 1522 for additional information.

Multimedia Material presented in a combination of text, graphics, video, animation, and sound, thereby “compressing” information into a more manageable and understandable format.

- N -

National Language Support (NLS) Support for multiple languages, multiple character set encodings and multiple ordering schemes in computer systems.

Navigation The act of traversing a collection of objects that are linked in some manner.

In DMA, navigation is the act of traversing from one independently persistent object to another related independently persistent object. Only some independently persistent DMA objects support navigation. Objects that do support navigation have a property, which has as its value, the OIID of the related independently persistent object.

An example of navigation is going from a container object to a relationship object. Another example is going from a relationship object to the contained object (i.e., a container object or a document version object).

NLS National Language Support.

NLS16 National Language Support with capabilities to handle 16bit code points.

- O -

Object Factory	A DMA object that provides the capability of creating new DMA objects of a specified class in the client's process space by implementing the <i>IdmaObjectFactory</i> interface. Typically the DMA Document Space and Scope objects act as object factories by supporting this interface.
Object Model	The collection of objects and interfaces and their relationship to each other that define a DMA system in total.
Object Instance Identifier	(OIID) A globally unique identifier used to identify and locate a persistent object in a DMA document space. Object Instance Identifiers are defined as DMA URL's.
Object-Valued Property	A property on a DMA object, whose value is another DMA object. Accessing a value of such a property yields an interface on that object.
ODBC	Open Database Connectivity.
ODMA	Open Document Management API.
OIID	Object Instance Identifier.
OIID Parser	A DMA Service Element that provides the capability to parse and extract components of a DMA OIID by implementing the <i>IdmaOIID</i> interface. Also, see DMA URL.
OLE	Microsoft's object technology for component software. It is a set of object-oriented system interfaces and services, which provides a framework to build reusable software components and is built on the foundation of the Component Object Model.

Open Database Connectivity (ODBC) A Microsoft-developed standard that defines a generic API for accessing relational databases. It implements ANSI SQL II. The API is based on work done by the SQL Access Group (SAG).

Open Document Management API (ODMA) An API for interfacing desktop applications to document management systems.

OpenDoc A set of standards, managed by the Component Integration Laboratories consortium, for document centric component software. See <http://www.cil.org/> for additional information.

- P -

Parent A container that contains other objects (children) by direct containment.

PDM Product Data Management.

Persistence The state of a DMA object that has been stored in a document space, such that it is accessible permanently until such time it is deleted.

Persistent Object	<p>A DMA object and its state that has been permanently stored in a document space until such time it is deleted. A persistent object may be either dependently or independently persistent.</p> <p>A <i>Dependently Persistent</i> object is one that represents sub-structure, and is owned by a larger independently persistent object. Therefore a dependently persistent object can not be saved directly; changes made to it are made persistent only when the owning independently persistent object is saved.</p> <p>An <i>Independently Persistent</i> object always has an OIID. In addition, if an object supports the <i>IdmaConnection</i> interface it is an independently persistent object, however all independently persistent objects are not required to support this interface (for example, the object may be read-only).</p> <p>The <i>IdmaConnection</i> interface has methods to make objects persistent in a document space.</p>
Point of Access	<p>A process space in which requests are made to DMA and DMA services via the DMA API. Typically this would be a DMA client application.</p>
Point of Presence	<p>An environment in which points-of-access and/or points-of-service is supported.</p>
Point of Service	<p>A process space in which DMA compliant service elements receive requests for operations via their DMA API compatible interfaces.</p>
Product Data Management	<p>(PDM) The management of product development-related data and documents.</p>
Proper Subclass	<p>A DMA class, which has been derived from another class that is not itself, is referred to as a proper subclass of the class from which it was derived from.</p> <p>Also, see subclass and immediate subclass.</p>
Property Model	<p>The primary means by which DMA object's expose and allow manipulation of their attributes (state). The <i>IdmaProperties</i> and <i>IdmaEditProperties</i> interfaces provide the operations to manipulate properties. Properties can be referred to either by a property ID or by an index, both of which can be discovered from an object's meta-data.</p>

- Q R -

Query	The top-level DMA object that is used in constructing and issuing a query. The Query object consists of the Query Root Object and other properties that affect the execution of the query. After a query is completely defined by constructing the Query Root and its sub-tree of objects, the Query Object is passed to the DMA ExecuteSearch method.
Query Model	The mechanism provided in DMA to express queries for searching document spaces. This is an object-based scheme with a rich set of capabilities that is also extensible, however it should be noted that it is dissimilar to textual query languages such as SQL.
Query Node	The base class of DMA objects used to form various components of a query including the Query Root object.
Query Result Row	The DMA objects produced as a result of enumerating a Query Result Set. Each Result Row object has the properties selected in the Selections list property of the main Query Root object.
Query Result Set	The DMA object generated as the primary output of executing a query. A result set object consists of zero or more Result Row objects that satisfy the search request issued.
Query Root	<p>A DMA object that is the root of a tree of dependent objects that specify a DMA query. The Query Root object tree may be thought of as a parse tree plus some other properties. If a query contains a sub-query, then one of the dependent objects is another Query Root object which itself is the root of another dependent tree of objects, etc.</p> <p>A Query Root object consists of object valued properties, which roughly correspond to the main clauses of a SQL style query language. These being the <i>From Expression</i>, <i>Selections</i>, <i>Query Expression</i> and the <i>Orderings</i>.</p> <p>All these objects are derived from the Query Node base class.</p>

Recovery	The process of restoring some data or an object to a previous state. Typically in the context of recovery of data from a backup in case of data loss or corruption.
Reference Middleware	An implementation of a DMA system manager and a sample system object that is owned and distributed by AIIM. This version of the middleware permits registration and access to a single document space within a single process space. It performs no coordination of query operations and performs no distributed operations.
Referential Containment	DMA model for representing containment, where a Container may contain multiple Containees by reference, and where a Containee may be contained in multiple Containers by reference. This essentially models an N: M relationship.
Relationship	The general mechanism provided in DMA to express an association between any two DMA objects. The Relationship class of objects and its sub-classes model various forms of associations between pairs of DMA objects. The Containment Relationship sub-class has been specifically defined in support of the DMA Containment Model.
Rendition	A DMA object that manages a particular representation (i.e. in a specific format, such as Word 8 or PostScript) of the content data of a document. Typically a document would have a rendition which normally is its native or editable format and one or more other renditions generated for printing, viewing and other purposes.
Repository Services	The management of files associated with a document in storage systems that have been integrated into the corporate computing environment. One of three functional areas for identifying and describing Document Management requirements.
Reservation	A DMA object defined in support of version management that carries information about a currently reserved or checked out versionable object.
Result Row	See Query Result Row.
Result Set	See Query Result Set.

Retrieval The process of copying documents or portions of documents from a Document Management system.

- S -

Scanning The process of converting paper or other hard copy into digital format.

Scope A DMA object through which a document space exposes details about its query capabilities and the searchable classes of objects contained in the document space. A DMA application must use a scope object to construct and initiate all search operations against a document space.

Also, see Merged Scope.

Scope Factory A DMA Service Element that implements the capability to merge multiple component scopes into a single logical scope for unified searching by supporting the *IdmaScopeFactory* interface.

Also, see Merged Scope.

Scratch Pad The term used to refer to DMA objects that are in a given process space. This is due to DMA using a model analogous to most editors to allow access to persistent objects in a document space, sometimes referred to as the scratch pad model.

DMA objects represent and provide means of modifying persistent state of objects in a document space. Therefore when a DMA object is first instantiated it is loaded with a snapshot of the persistent data which forms the scratch pad. The client can then manipulate the scratch pad without impacting other clients or the persistent form of the object in the document space until which time it decides to persist or discard the changes made in the scratch pad.

Scratch Pad Object Scratch Pad.

Search The mechanism of locating objects (Documents) by specifying contextual information or other constraints in the form of a query.

Searchable	<p>The concept of allowing DMA object classes to participate in a DMA query. This allows object instances of those classes to be located through a query.</p> <p>DMA has defined a minimum set of object classes that are required to be searchable. A document space provider must allow these and their sub-classes to be searchable in their implementation, however they may make additional DMA object classes be searchable.</p> <p>In contrast an object class that is not searchable can not participate in a DMA query and object instances of those classes can not be located via a DMA query.</p>
Search Model	<p>The uniform means by which DMA applications can locate documents managed by DMA document spaces irrespective of the underlying technologies, capabilities and organization of the document spaces. One of the most powerful aspects of this model is the ability to search across document spaces with a single query.</p>
Security	<p>The rules that restrict access to documents. Authentication refers to determining the identity of the user attempting the access; authorization refers to determining the set of privileges available to the user.</p>
Service Element	<p>A DMA compliant software system that offers access to document collections and other document management services by operating under the Service Provider Interface.</p>
Service Element Integration Interface	<p>The interface that supports registering and publishing a service element with a DMA compliant system and how a system implementation gets access to the service element.</p>
Service Integration Interface	<p>Service Element Integration Interface.</p>
Service Provider	<p>Service Element.</p>
Service Provider Interface	<p>(SPI) The combination of the Service Element Integration and the DMA COM interfaces (excluding the System Integration Interface). This can be thought of as the interface between a DMA service element and the DMA middleware layer.</p>
SGML	<p>Standard Generalized Markup Language.</p>

SPI	Service Provider Interface.
Standard Generalized Markup Language	(SGML) A markup language for defining the structure of documents generally used in technical publishing.
Storage Management	A system implemented to allow users to store and retrieve large numbers of documents.
Subclass	<p>A DMA class that has been derived from another class or itself is referred to as a subclass.</p> <p>Also, see proper subclass and immediate subclass.</p>
Superclass	<p>A DMA class that has been used to derive the definition of another class is referred to as the superclass of the derived class. In other words, the derived class may have been either directly or indirectly derived from its superclass. Therefore, all classes in the single inheritance hierarchy of a class, including its immediate superclass, are referred to as its superclasses.</p>
Synthetic Property	System-generated property.
System	A DMA System is the means by which a DMA compliant application gets access to a DMA service element. A DMA System can comprise one or more service elements. An attribute of a DMA system is a common character set encoding specified via a locale name.
System-Generated Property	(Synthetic Property) A property on a DMA object whose value has been or will be supplied by the document space implementation. Typically, these system-generated properties are read-only from a client's viewpoint. Object properties that have this attribute are indicated in the DMA specification.
System Integration Interface	The interface by which DMA client applications can locate and connect to DMA Systems and Service Elements. This interface is also used by system administration applications to manage registration of DMA compliant systems at a point of presence. The system manager implements the system integration interface for a particular operating environment.

System Integration Manager /
System Connection Manager /
System Manager

A DMA software component that encapsulates all of the operating environment dependencies and provides the means for service providers to register their systems and the means for client applications to discover and connect to registered systems. This component exposes the system integration interface for a given operating environment.

- T -

Technical Publishing Management

The management of the editorial and production processes leading to publication.

Threaded Versioning

A scheme of Versioning, either Linear or Branched, that allows a given version of an entity to occur in more than one list of versions or more than once in a single list of versions.

Translation

To change the storage format of a document on output.

- U V -

Universally Unique Identifier

(UUID) Defined by the Open Software Foundation (OSF) as part of the Distributed Computing Environment (DCE), it is a 128-bit integer value that is immutable and unique over all time and space. There is at least one well-defined algorithm to generate these values without the use of a central administration authority.

In COM these are referred to as Globally Unique Identifiers (GUID's) and are used to uniquely identify interfaces. DMA uses GUID's to uniquely identify properties, classes, query operators, collating sequences, etc. in addition to interfaces.

See OSF DCE home page for additional information.

URI

Universal Resource Identifier.

See <http://www.w3.org/> for additional information.

URL	<p>Universal Resource Locator. Defined as part of the Internet standards activities, it is a string representing a resource available on the Internet. Due to its extensibility and the rapid adoption of Internet standards, URL's have wide spread applicability and have become common place.</p> <p>DMA uses URL's for persistent object identification and location. See DMA URL's.</p> <p>See http://www.w3.org/ for additional information.</p>
URN	<p>Universal Resource Name.</p>
UUID	<p>Universally Unique Identifier.</p>
Version	<p>An instance of an object at a particular point in its progression. Typically versions of documents are created explicitly by the user performing an operation such as check-in.</p>
Version Control	<p>A Document Management feature, whereby multiple versions of a document (which can be created after repeated check-ins) are managed.</p>
Version Description	<p>An object used in support of DMA Versioning. Version description objects carry information (also referred to as "edge data") about the connection between the version series and the versionable object.</p>
Version Series	<p>An object used in support of DMA Versioning. The version series object carries the configuration of a single "line" of versions that are viewed as generally having a progressive history. Version series are made up of a linear sequence of zero or more version description objects. The version series is part of exactly one configuration history.</p>
Versionable	<p>Versionable Object.</p>
Versionable Object	<p>A DMA object that is allowed to be Versioned (i.e. be part of a Version Series). Versionable objects inherit from the DMA Versionable class.</p>

Versioning Model	The mechanism provided in DMA to allow a document space to present and deliver version management functionality to a DMA application through a well-defined set of objects.
------------------	---

View	The process of displaying the contents of a document in human-readable form.
------	--

- W X Y Z -

W3C	World Wide Web Consortium.
-----	----------------------------

WfMC	Workflow Management Coalition.
------	--------------------------------

Workflow	Refers to automating group business processes by sequencing tasks and routing information based on business rules and the roles people play in the process.
----------	---

Workflow Management Coalition	(WfMC) An international coalition of workflow vendors, users and analysts with a mission to promote the use of workflow through the establishment of standards for software terminology, interoperability and connectivity between workflow products. See WfMC home page for additional information.
-------------------------------	---

World Wide Web	(WWW) A global distributed information system on the Internet, organized as a seamless world in which information from any source can be accessed easily and consistently. See http://www.w3.org/ for additional information.
----------------	---

World Wide Web Consortium	(W3C) An industry consortium which develops common standards for the evolution of the Web by producing specifications and reference software. See http://www.w3.org/ for additional information.
---------------------------	--

WWW	World Wide Web.
-----	-----------------