



1  
2  
3  
4

**Document Number: DSP0252**

**Date: 2010-04-22**

**Version: 1.0.1**

5 **Configuration Management Database (CMDB)**  
6 **Federation Specification**

7 **Document Type: Specification**  
8 **Document Status: DMTF Standard**  
9 **Document Language: E**

10

11 Copyright Notice

12 Copyright © 2009-2010 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

13 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems  
14 management and interoperability. Members and non-members may reproduce DMTF specifications and  
15 documents, provided that correct attribution is given. As DMTF specifications may be revised from time to  
16 time, the particular version and release date should always be noted.

17 Implementation of certain elements of this standard or proposed standard may be subject to third party  
18 patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations  
19 to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose,  
20 or identify any or all such third party patent right, owners or claimants, nor for any incomplete or  
21 inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to  
22 any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize,  
23 disclose, or identify any such third party patent rights, or for such party's reliance on the standard or  
24 incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any  
25 party implementing such standard, whether such implementation is foreseeable or not, nor to any patent  
26 owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is  
27 withdrawn or modified after publication, and shall be indemnified and held harmless by any party  
28 implementing the standard from any and all claims of infringement by a patent owner for such  
29 implementations.

30 For information about patents held by third-parties which have notified the DMTF that, in their opinion,  
31 such patent may relate to or impact implementations of DMTF standards, visit  
32 <http://www.dmtf.org/about/policies/disclosures.php>.

33

# CONTENTS

35	Foreword .....	5
36	Introduction .....	7
37	1 Scope .....	11
38	2 Normative References.....	11
39	3 Terms and Definitions .....	12
40	3.1 Requirements Terms.....	12
41	3.2 Background Terminology.....	13
42	4 Symbols and Abbreviated Terms.....	15
43	5 Architecture .....	15
44	5.1 Overview .....	15
45	5.2 Roles.....	16
46	5.3 Services Overview .....	17
47	5.4 Identity Reconciliation.....	18
48	5.5 Data Elements Overview .....	19
49	6 Query Service.....	22
50	6.1 Overview .....	22
51	6.2 GraphQL Operation Outline.....	22
52	6.3 Content Selection.....	25
53	6.4 Constraints.....	28
54	6.5 XPath Expressions and Normalization .....	34
55	6.6 GraphQL Response.....	36
56	6.7 GraphQL Faults .....	39
57	7 Registration Service .....	41
58	7.1 Overview .....	41
59	7.2 Register.....	42
60	7.3 Deregister .....	47
61	8 Service Metadata .....	49
62	8.1 Overview .....	49
63	8.2 Common Service Metadata Elements .....	50
64	8.3 queryServiceMetadata .....	52
65	8.4 registrationServiceMetadata .....	56
66	ANNEX A (normative) URIs and XML Namespaces .....	57
67	ANNEX B (normative) CMDB Federation XSD and WSDL .....	58
68	ANNEX C (normative) Fault Binding to SOAP.....	59
69	ANNEX D (informative) Query Examples .....	61
70	D.1 GraphQL Example 1 .....	61
71	D.2 GraphQL Example 2 .....	63
72	ANNEX E (informative) Detailed UML Class Diagrams.....	68
73	ANNEX F (informative) Sample WSDL Binding.....	69
74	ANNEX G (informative) Change Log .....	73
75	Bibliography .....	74
76		

77 **Figures**

78 Figure 1 – CMDB as the Foundation for IT Management Processes..... 7  
 79 Figure 2 – Example Aggregate View from a Federated CMDB..... 8  
 80 Figure 3 – CMDB Roles and Services ..... 16  
 81 Figure 4 – Identity Reconciliation..... 18  
 82 Figure 5 – Data and Services Overview ..... 19  
 83 Figure 6 – Record Type Extension Examples..... 52  
 84

85 **Tables**

86 Table 1 – Service Usage Patterns ..... 18  
 87 Table 2 – Operators Supported for XSD Built-in Datatypes ..... 31  
 88 Table C-1 – [Code] Properties ..... 59  
 89 Table D-1 – "User (ContactInfo)" Data..... 63  
 90 Table D-2 – "Computer (ComputerConfig)" Data..... 64  
 91 Table D-3 – "Administers" Data ..... 64  
 92

93

## Foreword

94 The *Configuration Management Database (CMDB) Federation Specification* (DSP0252) was prepared by  
95 the CMDB Federation Working Group.

96 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems  
97 management and interoperability.

## 98 Acknowledgements

99 The CMDB Federation Working Group wishes to acknowledge the following people.

100 Authors:

- 101 • Forest Carlisle – CA
- 102 • Jacob Eisinger – IBM
- 103 • Mark Johnson – IBM (Editor)
- 104 • Vincent Kowalski – BMC Software
- 105 • Jishnu Mukerji – HP
- 106 • David Snelling – Fujitsu
- 107 • William Vambenepe – Oracle
- 108 • Marv Waschke – CA
- 109 • Van Wiles – BMC Software

## 110 Conventions

111 This specification uses the following syntax to define outlines for messages:

- 112 • The syntax appears as an XML instance, but values in italics indicate data types instead of  
113 literal values.
- 114 • The following characters are appended to elements and attributes to indicate cardinality:
  - 115 – "?" (0 or 1)
  - 116 – "\*" (0 or more)
  - 117 – "+" (1 or more)
  - 118 – The absence of any of the above characters indicates the default (exactly 1).
- 119 • The character "|" is used to indicate a choice between alternatives.
- 120 • The characters "(" and ")" are used to indicate that contained items are to be treated as a group  
121 with respect to cardinality or choice.
- 122 • The characters "[" and "]" are used to call out references and property names.
- 123 • xs:any and xs:anyAttribute indicate points of extensibility. Additional children or attributes may  
124 be added at the indicated extension points but shall not contradict the semantics of the parent  
125 owner, respectively. By default, if a receiver does not recognize an extension, the receiver  
126 should ignore the extension; exceptions to this processing rule, if any, are clearly indicated  
127 below.

- 128 • Ellipses (that is, "...") indicate that details are omitted for simplicity, and a further explanation is  
129 provided below.
- 130 • XML namespace prefixes are used to indicate the namespace of the element being defined or  
131 referenced.

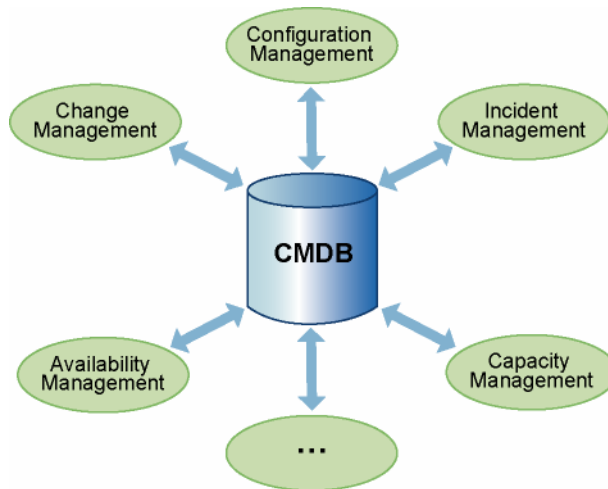
132

## Introduction

133 Many organizations are striving to base IT management on a Configuration Management Database  
 134 (CMDB). A CMDB contains data describing the following entities:

- 135 • managed resources, such as computer systems and application software
- 136 • process artifacts, such as incident, problem, and change records
- 137 • relationships among managed resources and process artifacts

138 The contents of the CMDB should be managed by a configuration management process and serve as the  
 139 foundation for other IT management processes, such as change management and availability  
 140 management, as shown in Figure 1.



141

142 **Figure 1 – CMDB as the Foundation for IT Management Processes**

143 However, in practice it is challenging to implement such a CMDB because the management data are  
 144 scattered across repositories that are poorly integrated or coordinated.

145 The definition of a CMDB in the context of this specification is based on the definition described in the IT  
 146 Infrastructure Library (ITIL): a database that tracks and records configuration items associated with the IT  
 147 infrastructure and the relationships between them. Strictly speaking, the ITIL CMDB contains a record of  
 148 the expected configuration of the IT environment, as authorized and controlled through the change  
 149 management and configuration management processes. The federated CMDB in this specification  
 150 extends this base definition to federate any management information that complies with the specification's  
 151 patterns, schema, and interfaces, such as the discovered actual state in addition to the expected state.  
 152 Typically, an administrator selects the data to be included in a CMDB by configuring the tool that  
 153 implements the CMDB.

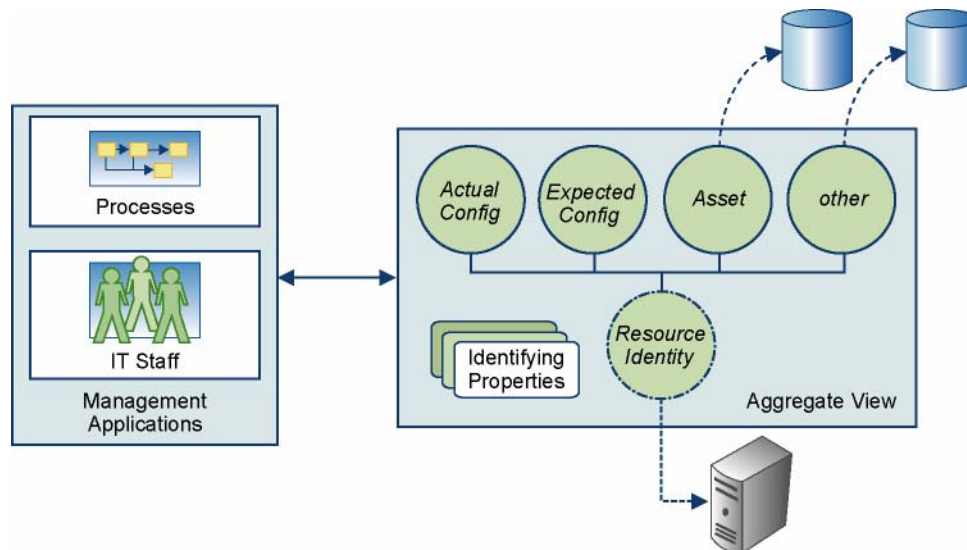
154 The federated CMDB described in this specification is a collection of services and data repositories that  
 155 contain configuration and other data records about resources. The term "resource" includes configuration  
 156 items (for example, a computer system, an application, or a router), process artifacts (for example, an  
 157 incident record or a change record), and relationships between configuration items and process artifacts.  
 158 The architecture describes a logical model and does not necessarily reflect a physical manifestation.

159 **Objectives**

160 This section describes the functionality and target IT environment that this specification supports.

161 **Functionality**

162 The federated CMDB that would result from using this specification would provide a single aggregate  
 163 view of the data about an IT resource, even if the data is from different heterogeneous data repositories,  
 164 as shown in Figure 2. Clients, such as IT processes, management applications, and IT staff would use a  
 165 query service defined in the specification to access aggregated or non-aggregated views. Data  
 166 repositories would use the services described in the specification to provide the aggregated view.



167

168 **Figure 2 – Example Aggregate View from a Federated CMDB**

169 The federated CMDB could support the following scenarios. (However, the scenarios that a federated  
 170 CMDB supports are left entirely to the discretion of each implementation.)

- 171
- 172 • Maintain an accurate picture of IT inventory from a combination of asset information (finance)  
and deployment/configuration information
  - 173 • Reflect changes to IT resources, including asset and licensing data, across all repositories and  
174 data sources
  - 175 • Compare expected configuration versus actual configuration
  - 176 • Enable version awareness, such as in the following examples:  
177 – Coordinate planned configuration changes  
178 – Track change history
  - 179 • Relate configuration and asset data to other data and data sources, such as incident, problem,  
180 and service levels. The following are some examples:  
181 – Integration of change management and incident management with monitoring information  
182 – SLA incident analysis, by using the service desk and incident information in a dependency  
183 analysis on both configurations and change records



## 184 Target IT Environment

185 This specification is intended to address requirements in IT environments that have the following  
186 characteristics:

- 187 • There are strong requirements to consolidate into one or more databases (logical or physical) at  
188 least some key data from the many management data repositories so that IT processes can be  
189 more effective and efficient.
- 190 • IT organizations are diverse in terms of their existing tools, process maturity level, usage  
191 patterns, and preferred adoption models.
- 192 • There are several (and possibly many) management data repositories (MDRs), each of which  
193 may be considered an authoritative source for some set of data.
- 194 • The authoritative data for a resource may be dispersed across multiple MDRs.
- 195 • It is often neither practical nor desirable for all management data to be kept in one data  
196 repository, though it may be practical and desirable to consolidate various subsets of the data  
197 into fewer databases.
- 198 • Existing management tools will often continue to use their existing data sources. Only after an  
199 extended period of time would it be realistic to expect all of the existing management tools to be  
200 modified to require and utilize new consolidated databases.

## 201 Out-of-Scope Implementation Details

202 The following implementation details are outside the scope of this specification:

- 203 • The mechanisms used by each management data repository to acquire data. For example, the  
204 mechanisms could be external instrumentation or proprietary federation and replication function.
- 205 • The mechanisms and formats used to store data. The specification is concerned only with the  
206 exchange of data. A possible implementation is a relational database that stores data in tables.  
207 Another possible implementation is a front-end that accesses the data on demand from an  
208 external provider, similar to a commonly used CIMOM/provider pattern.
- 209 • The processes used to maintain the data in the federated CMDB. The goal of the specification  
210 is to enable IT processes to manage this data, but not to require or dictate specific processes.
- 211 • The mechanisms used to change the actual configuration of the IT resources and their  
212 relationships. The goal of the specification is to provide the means to represent changes as or  
213 after they are made, but not to be the agent that makes the change.

## 214 Technological Assumptions

215 This specification is based on some assumptions with regard to underlying technology and the context of  
216 computing standards that exist at the time of its writing.

### 217 Underlying Technology

218 The technologies behind CMDBs include Web Services and database management systems.

### 219 Web Services

220 Although the interface specification contained herein is generic, it assumes that implementations will be  
221 based on Web Services. Although interfaces based on programming languages such as Java and C#  
222 could be derived from this specification, such interfaces are considered out of scope and are not  
223 addressed here.

**224 Database Management Systems**

225 In general practice CMDBs are implemented using commercially available database technology. Although  
226 this specification is about how one or more CMDBs federate data using a standard mechanism, no  
227 assumptions are made about how that federated data is stored or persisted. The specification focuses on  
228 the interfaces; their behavior, and the data types they convey. Database technology is clearly a needed  
229 component in the implementation of this specification, but its use is considered to be a hidden detail of  
230 such implementations.

# 231 Configuration Management Database (CMDB) Federation 232 Specification

## 233 1 Scope

234 This specification describes the architecture and interactions for federating data repositories together to  
235 behave as a data store that satisfies the role of a Configuration Management Database (CMDB), or as  
236 the federated repository that is the heart of a Configuration Management System, as described in the ITIL  
237 best practices, version 3. For brevity, the remainder of the document uses the term CMDB, even when  
238 the term Configuration Management System would be at least as appropriate. The federation provides an  
239 aggregate view of a resource, even though the data and underlying repositories are heterogeneous. A  
240 query interface is defined for external clients to access these data.

## 241 2 Normative References

242 The following referenced documents are indispensable for the application of this document. For dated  
243 references, only the edition cited applies. For undated references, the latest edition of the referenced  
244 document (including any amendments) applies.

245 IETC RFC 2616, *Hypertext Transfer Protocol – HTTP/1.1*, June 1999,  
246 <http://www.ietf.org/rfc/rfc2616.txt>

247 ISO 8601, Third edition, 2004-12-01, *Data elements and interchange formats — Information interchange*  
248 *— Representation of dates and times*

249 ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*,  
250 <http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype>

251 ITSMF, *ITIL Version 3 Glossary of Terms and Definitions*, May 2007,  
252 [http://www.itsmf.co.uk/web/FILES/Publications/ITILV3\\_Glossary\\_English\\_v1\\_2007.pdf](http://www.itsmf.co.uk/web/FILES/Publications/ITILV3_Glossary_English_v1_2007.pdf)

253 W3C, *Simple Object Access Protocol (SOAP) 1.1*, May 2000,  
254 <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

255 W3C, *SOAP Version 1.2 Part 1: Messaging Framework*, April 2007,  
256 <http://www.w3.org/TR/2006/REC-xml-20060816/>

257 W3C, *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, September 2006,  
258 <http://www.w3.org/TR/2006/REC-xml-20060816/>

259 W3C, *XML Schema 1.0 Part 1: Structures (Second Edition)*, October 2004,  
260 <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>

261 W3C, *XML Schema 1.0 Part 2: Datatypes (Second Edition)*, October 2004,  
262 <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>

263 W3C, *XML Path Language (XPath) 1.0*, November 1999,  
264 <http://www.w3.org/TR/1999/REC-xpath-19991116>

265 W3C, *XML Path Language (XPath) 2.0*, January 2007,  
266 <http://www.w3.org/TR/2007/REC-xpath20-20070123/>

267 W3C, *XQuery 1.0 and XPath 2.0 Functions and Operators*, January 2007, [http://www.w3.org/TR/xquery-](http://www.w3.org/TR/xquery-operators/)  
268 [operators/](http://www.w3.org/TR/xquery-operators/)

269 W3C, *XSLT 2.0 and XQuery 1.0 Serialization*, January 2007, [http://www.w3.org/TR/xslt-xquery-](http://www.w3.org/TR/xslt-xquery-serialization/)  
270 [serialization/](http://www.w3.org/TR/xslt-xquery-serialization/)

271 W3C, *Web Services Description Language (WSDL) 1.1*, March 2001, [http://www.w3.org/TR/2001/NOTE-](http://www.w3.org/TR/2001/NOTE-wsdl-20010315)  
272 [wsdl-20010315](http://www.w3.org/TR/2001/NOTE-wsdl-20010315)

273 ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*,  
274 <http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype>

## 275 **3 Terms and Definitions**

276 For the purposes of this document, the following terms and definitions apply.

### 277 **3.1 Requirements Terms**

#### 278 **3.1.1**

##### 279 **can**

280 used for statements of possibility and capability, whether material, physical, or causal

#### 281 **3.1.2**

##### 282 **cannot**

283 used for statements of possibility and capability, whether material, physical or causal

#### 284 **3.1.3**

##### 285 **conditional**

286 indicates requirements to be followed strictly in order to conform to the document when the specified  
287 conditions are met

#### 288 **3.1.4**

##### 289 **mandatory**

290 indicates requirements to be followed strictly in order to conform to the document and from which no  
291 deviation is permitted

#### 292 **3.1.5**

##### 293 **may**

294 indicates a course of action permissible within the limits of the document

#### 295 **3.1.6**

##### 296 **need not**

297 indicates a course of action permissible within the limits of the document

#### 298 **3.1.7**

##### 299 **optional**

300 indicates a course of action permissible within the limits of the document

#### 301 **3.1.8**

##### 302 **shall**

303 indicates requirements to be followed strictly in order to conform to the document and from which no  
304 deviation is permitted

305 **3.1.9**306 **shall not**

307 indicates requirements to be followed strictly in order to conform to the document and from which no  
308 deviation is permitted

309 **3.1.10**310 **should**

311 indicates that among several possibilities, one is recommended as particularly suitable, without  
312 mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

313 **3.1.11**314 **should not**

315 indicates that a certain possibility or course of action is deprecated but not prohibited

316 **3.2 Background Terminology**

317 This section defines terms used throughout this specification. For the most part, these terms are adopted  
318 from other sources. The terms are defined here to clarify their usage in this specification and, in some  
319 cases, to show their relationship to the use of the terms in other sources. In particular, this specification  
320 shares concepts with Information Technology Infrastructure Library (ITIL). ITIL is not a standard and does  
321 not provide normative definitions of terms. However, the ITIL version 3 glossary is quoted below as  
322 representative of the ITIL position.

323 **3.2.1**324 **configuration item**325 **CI**

326 a basic tangible or intangible entity in a configuration management solution such as a CMDB.

327 ITIL version 3 defines a CI as follows:

328 "Any Component that needs to be managed in order to deliver an IT Service. Information about  
329 each CI is recorded in a Configuration Record within the Configuration Management System  
330 and is maintained throughout its Lifecycle by Configuration Management. CIs are under the  
331 control of Change Management. CIs typically include IT Services, hardware, software, buildings,  
332 people, and formal documentation such as Process documentation and SLAs."

333 **3.2.2**334 **configuration management database**335 **CMDB**

336 ITIL defines a CMDB as follows:

337 "A database used to store Configuration Records throughout their Lifecycle. The Configuration  
338 Management System maintains one or more CMDBs, and each CMDB stores *Attributes* of CIs,  
339 and Relationships with other CIs."

340 A configuration management database (CMDB) is often implemented using standard database  
341 technology and typically persists CI lifecycle data as records (or configuration records) in that database.  
342 Configuration records are managed according to some data or information model of the IT environment.  
343 One of the goals of this specification is to expedite the federated implementation of multiple CMDBs in a  
344 single configuration management system.

345 **3.2.3**346 **configuration management system**347 **CMS**

348 ITIL defines (in part) a configuration management system as follows:

349 "A set of tools and databases that are used to manage an IT Service Provider's Configuration  
350 data. The CMS also includes information about Incidents, Problems, Known Errors, Changes

351 and Releases; and may contain data about employees, Suppliers, locations, Business Units,  
352 Customers and Users."

353 A configuration management system is presumed to be a federation of CMDBs and other management  
354 data repositories. The federated CMDB described in this specification is a good match with the database  
355 requirements of a configuration management system.

### 356 **3.2.4** 357 **configuration record**

358 ITIL defines a configuration record as follows:

359 A Record containing the details of a Configuration Item. Each Configuration Record documents  
360 the Lifecycle of a single CI. Configuration Records are stored in a Configuration Management  
361 Database.

362 For the purposes of this specification, a CI is a tangible or intangible entity treated in the abstract by this  
363 specification, while a configuration record contains concrete data pertaining to a CI. More than one  
364 configuration record may be associated with a given CI. Often configuration records will be from different  
365 data sources or document different points in the lifecycle of a CI. It is possible for configuration records  
366 associated with a single CI to contain data that may appear contradictory and require mediation.

### 367 **3.2.5** 368 **federated CMDB**

369 a combination of multiple management data repositories (MDRs), at least one of which federates the  
370 others, into an aggregate view of management data.

371 NOTE: Whereas "federated CMDB" refers to the combination of all the data repositories, "federating CMDB" is a  
372 specific role performed by a data repository that federates other MDRs.

### 373 **3.2.6** 374 **federation**

375 the process of combining information from management data repositories (MDRs) into a single  
376 representation that can be queried in a consistent manner. Federation is often contrasted with extract,  
377 transform, and load (ETL) systems which transfer and store data from one repository to another. This  
378 specification does not exclude ETL activities, especially for caching, but the main purpose of the  
379 specification is to support systems that minimize or eliminate transferring and storing data from MDRs in  
380 federators.

### 381 **3.2.7** 382 **graph**

383 a kind of data structure, specifically an abstract data type, that consists of a set of nodes and a set of  
384 edges that establish relationships (connections or links) between the nodes. In this specification the  
385 nodes are items and the edges are relationships.

### 386 **3.2.8** 387 **identity**

388 a set of qualities or characteristics that distinguish an entity from other entities of the same or different  
389 types. This set of qualities may be called the "identifying properties" of the real world entity for which the  
390 CMDB contains data.

### 391 **3.2.9** 392 **Information Technology Infrastructure Library** 393 **ITIL**

394 a framework of best practices for delivering IT services. Two versions of ITIL are commonly in use:  
395 version 2 released in 2000 and version 3 released in 2007. Because ITIL version 3 has not yet  
396 superseded version 2 in practice, both versions have been considered in preparing this specification. A  
397 CMDB is a key component in the ITIL best practices.

## 398 **4 Symbols and Abbreviated Terms**

### 399 **4.1**

#### 400 **CI**

401 configuration item

### 402 **4.2**

#### 403 **CMDB**

404 configuration management database

### 405 **4.3**

#### 406 **CMDBf**

407 configuration management database federation

### 408 **4.4**

#### 409 **CMS**

410 configuration management system

### 411 **4.5**

#### 412 **ITIL**

413 Information Technology Infrastructure Library

### 414 **4.6**

#### 415 **MDR**

416 management data repository

### 417 **4.7**

#### 418 **SACM**

419 service asset and configuration management

### 420 **4.8**

#### 421 **SLA**

422 service level agreement

### 423 **4.9**

#### 424 **WSDL**

425 Web Service Definition Language

## 426 **5 Architecture**

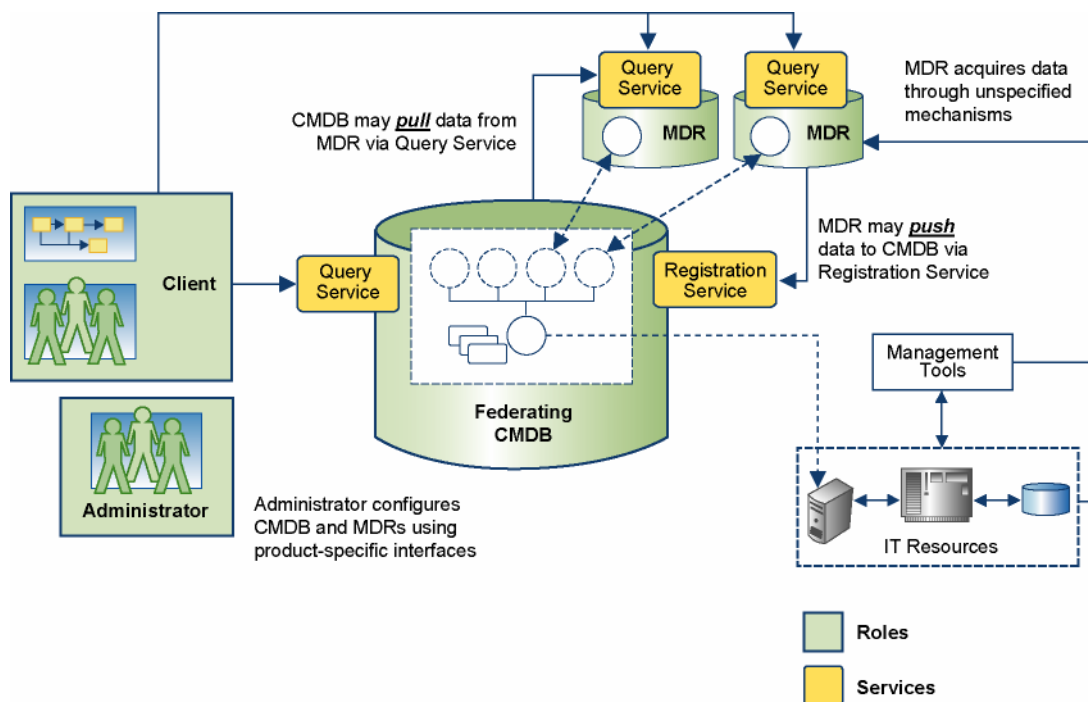
### 427 **5.1 Overview**

428 As shown in Figure 3, the architecture defines the following four roles:

- 429 • management data repository
- 430 • federating CMDB
- 431 • client
- 432 • administrator

433 These roles implement or use the following two services:

- 434 • Query Service
- 435 • Registration Service



436

437 **Figure 3 – CMDB Roles and Services**

438 **5.2 Roles**

439 **5.2.1 Management Data Repository (MDR)**

440 An MDR provides data about managed resources (for example, computer systems, application software,  
 441 and buildings), process artifacts (for example, incident records and request for change forms), and the  
 442 relationships between them. In this architecture, managed resources and process artifacts are both called  
 443 "items". The means by which the MDR acquires data is not specified, but the means can include acquiring  
 444 data directly from instrumented resources or indirectly through management tools.

445 Each MDR has an ID that is unique within (at least) a group of federated MDRs, and preferably globally  
 446 unique.

447 **5.2.2 Federating CMDB**

448 A federating CMDB is an MDR with additional capabilities. It federates data from MDRs; it may also  
 449 contain non-federated data. It provides an aggregate view of an item or relationship, potentially using data  
 450 from multiple MDRs. A federating CMDB and all the MDRs together comprise a federated CMDB.

451 It is possible for one federating CMDB to have its data federated by a second federating CMDB. In this  
 452 case, the first federating CMDB would appear to the second federating CMDB to be an MDR. The second  
 453 federating CMDB would not be aware of any federation performed by the first federating CMDB.



### 454 5.2.3 Client

455 A client is a consumer of management data, either directly from an MDR or through an aggregated view  
456 from a federating CMDB. Examples of clients are IT process workflows, management tools, and IT  
457 administrators. Clients only read data; there are no provisions for a client to update data through an  
458 interface defined in this architecture.

### 459 5.2.4 Administrator

460 An administrator configures MDRs and federating CMDBs so they can interact with each other.  
461 Administration includes selecting and specifying the data that is federated, describing service endpoints,  
462 and describing which data are managed through each endpoint. Administration is done using interfaces  
463 not defined in this architecture and that may be specific to each tool that acts in the MDR or federating  
464 CMDB role.

## 465 5.3 Services Overview

466 The subsequent clauses explain service types, federation modes, and service usage patterns.

### 467 5.3.1 Service Types

468 The architecture defines two services: Query Service and Registration Service. A service has an  
469 implementor and a client (caller).

#### 470 5.3.1.1 Query Service

471 Both MDRs and federating CMDBs may implement the Query Service to make data available to Clients.  
472 Queries may select and return items, relationships, or graphs containing items and relationships, and the  
473 data records associated with each item and relationship. An MDR or a federating CMDB may declare the  
474 data record types that its Query Service supports.

#### 475 5.3.1.2 Registration Service

476 A federating CMDB may implement the Registration Service. An MDR may call the Registration Service  
477 to register data that it has available for federation. A federating CMDB may declare the data types that its  
478 Registration Service supports. An MDR maps its data to the supported types.

### 479 5.3.2 Federation Modes

480 The two modes available to federate data are push mode and pull mode. A federating CMDB shall use at  
481 least one mode and may use both.

#### 482 5.3.2.1 Push Mode

483 In push mode, the MDR initiates the federation. Typically an administrator configures the MDR by  
484 selecting to federate some data types that are supported by both the MDR and the Registration Service.  
485 The MDR notifies the Registration Service any time this data is added, updated, or deleted. Depending on  
486 the extent of the data types, the registered data may be limited to identification data or it may include  
487 other properties that describe the item or relationship state.

#### 488 5.3.2.2 Pull Mode

489 In pull mode, the federating CMDB initiates the federation. Typically, an administrator configures the  
490 federating CMDB by selecting the MDR data types that will be federated. The federating CMDB queries  
491 MDRs for instances of this data. Depending on the implementation, the federating CMDB may pass  
492 through queries to MDRs without maintaining any state, or it may cache some set of MDR data, such as  
493 the data used to identify items and relationships.

5.3.3 Service Usage Patterns

Table 1 lists the service usage patterns for the roles described in 5.2 that implement or use the services.

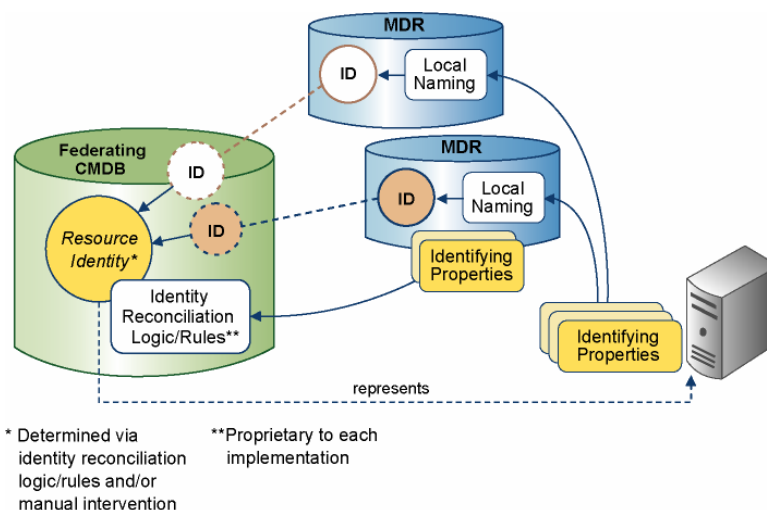
Table 1 – Service Usage Patterns

Pattern (Role + Mode)	Query Service		Registration Service	
	Implementation	Client	Implementation	Client
Federating CMDB – Push Mode	Required	Optional	Required	N/A
Federating CMDB – Pull Mode	Required	Required	N/A	N/A
MDR – Push Mode	Optional	N/A	N/A	Required
MDR – Pull Mode	Required	N/A	N/A	N/A
Client (external)	N/A	Required	N/A	N/A

5.4 Identity Reconciliation

Managed resources are often identified in multiple ways, depending on the management perspective. Examples of management perspectives are a change management process and an availability monitoring tool. Understanding how to identify resources, and reconciling the identifiers across multiple perspectives, is an important capability of a federating CMDB. The following pattern is typically used for identity reconciliation:

- Each MDR identifies a resource based on one or more identifying properties of the resource. Identifying properties are physical or logical properties that distinguish unique instances of resources. Examples are MAC addresses, host names, and serial numbers. Often, more than one property will be necessary to uniquely distinguish a resource, especially when information is incomplete. In addition, when two or more MDRs contain data about a single resource, individual MDRs may choose or have available different identifying properties, which they may use in their resource identifier for the item or relationship.
- Each MDR knows at least one unique and unambiguous identifier for each item or relationship it contains or provides access to through the Query Service.
- A federating CMDB attempts to reconcile the item and relationship identification information from each MDR, recognizing when they refer to the same item or relationship.



514

515

Figure 4 – Identity Reconciliation

516 The federating CMDB performs this identity mapping using any combination of automated analysis and  
 517 manual input, as shown in Figure 4. In a typical implementation the federating CMDB analyzes the  
 518 identifying properties to determine the resource identity. As each item or relationship is registered, the  
 519 service determines if this item or relationship is already registered or is new. The determination of identity  
 520 is seldom absolute and often must rely on heuristics because different MDRs typically know about  
 521 different characteristics of an entity and thus establish different sets of identifying properties that  
 522 characterize the entities they handle. Further, the determination may change as additional information is  
 523 discovered and MDRs add, subtract, or change identifying properties as systems evolve.

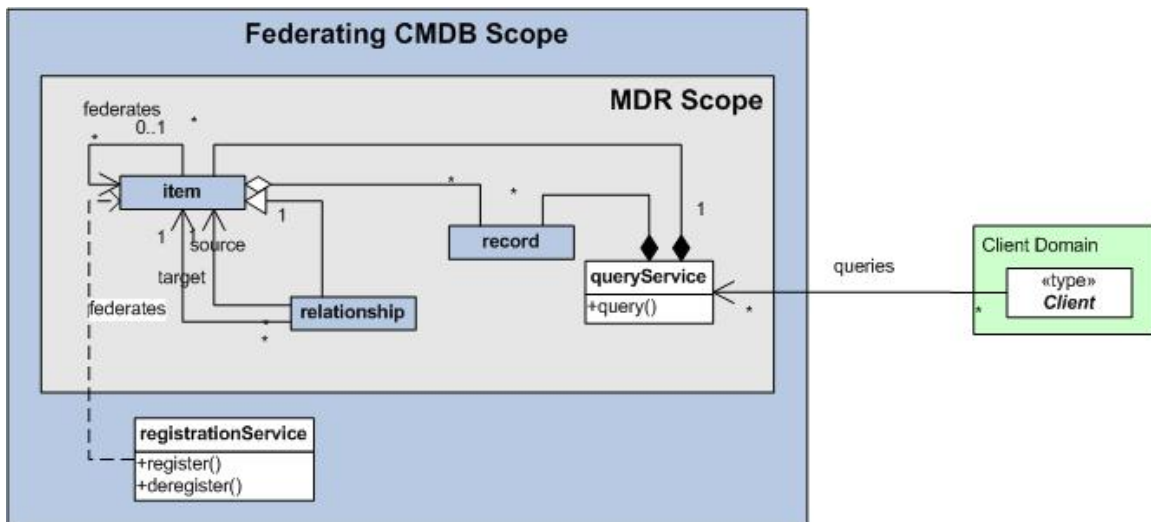
524 **5.5 Data Elements Overview**

525 Subsequent clauses provide an overview of the elements used to organize the data in MDRs and  
 526 federating CMDBs.

527 **5.5.1 Managed Data**

528 The architecture defines three elements that organize the data that repositories exchange: item,  
 529 relationship, and record.

530 The data contained in an MDR or federating CMDB is a graph where the items are nodes and the  
 531 relationships are links. The graph is not necessarily connected. (In other words, there may not be a  
 532 relationship trail from any item to any other item.) The query interface described below allows queries to  
 533 be constructed based on aspects of the graph (for example, existence of a relationship between two  
 534 items) and based on properties of the items and relationships (for example, requirements for a certain  
 535 value of a given record property or a certain type for the item and relationship).



536

537 **Figure 5 – Data and Services Overview**

538 **5.5.1.1 Item**

539 An item represents a managed resource (for example, computer systems, application software, and  
 540 buildings) or a process artifact (for example, an incident record and request for change form). With this  
 541 definition, "item" is a superset of the "configuration item" term defined in ITIL. Formally:

- 542 • Each item shall have at least one ID that is unique within the scope of the MDR that contains it  
 543 and that serves as a key.
- 544 • After an ID has been assigned to an item, it may be used in any situation requiring an ID.

- 545 • After an ID has been assigned to an item, it shall never refer to anything except the original  
546 item.
- 547 • An instance ID of an item is the composition of the unique MDR ID and the unique item ID  
548 assigned by that MDR. The instance ID is therefore unique within the group of federated  
549 repositories.

550 Examples of when an item might have multiple IDs include when an item is reconciled across several  
551 MDRs and the federating CMDB knows it by all of the IDs that have been assigned by different MDRs;  
552 when two items are thought to be different but are later reconciled to the same item; or when an ID  
553 changes for any other reason.

554 Given that each MDR has a unique ID within the group of federated repositories, and that each MDR  
555 assigns a unique ID within its own scope, the combination of the MDR ID and the MDR-assigned item ID  
556 results in an instance ID that is unique within the group of federated repositories. This instance ID serves  
557 two purposes:

- 558 • It is an unambiguous identifier for the representation of the item held by the MDR that assigned  
559 the instance ID.
- 560 • The MDR ID portion of the instance ID identifies the MDR that assigned the instance ID. A client  
561 may introspect the instance ID to extract the MDR ID. The client may then use the MDR ID to  
562 acquire the Query Service address for this MDR. For example, the MDR ID might be the key in  
563 a registry that contains the service addresses for each MDR. The client may then issue a query  
564 to this address to retrieve the representation of the item.

### 565 5.5.1.2 Relationship

566 A relationship represents a connection from a source item to a target item. Examples include software  
567 "runs" on an operating system, an operating system is "installed" on a computer system, an incident  
568 record "affects" a computer system, and service "uses" (another) service. Relationships have the  
569 following characteristics:

- 570 • A relationship links exactly two items, one the source and one the target, and provides  
571 information pertaining to that relationship.
- 572 • A relationship is a subclass of an item (though the relationship XML schema does not formally  
573 extend the item XML schema), and has all the characteristics of an item. For example, each  
574 relationship shall have an ID that is unique within the scope of the MDR that contains it and that  
575 serves as a key, and a reconciled relationship may have more than one ID.

#### 576 5.5.1.2.1 Relationship Roles

577 The two endpoints of a relationship are not equivalent. In the general case, items at these endpoints play  
578 different roles in the relationship. Some relationships may not have any such semantic distinction  
579 because they are symmetrical (e.g. "sibling"), but this is not the general case. An example of the general  
580 case is an "employment" relationship which links an "employer" to an "employee".

581 CMDBf designates the endpoints as "source" and "target" to distinguish them. There are no semantics  
582 attached to these terms, other than a convention that when a relationship is represented graphically by an  
583 arrow, the arrow goes from the source to the target. The relationship record type (see 5.5.1.3)  
584 documentation should describe the role semantics of the "source" and "target" endpoints.

#### 585 5.5.1.3 Record

586 A record contains properties that describe an item or relationship. Records have the following  
587 characteristics:

- 588 • A record is associated with exactly one item or relationship.

589       • A record may contain properties that are useful to identify the item or relationship, or it may  
590       contain other properties that describe the item or relationship.

591       • Several records, possibly of various types, may be associated with the same item or  
592       relationship.

593       Records may differ from other records for various reasons, including types of data (for example, asset  
594       versus configuration), different sets of properties from different providers, different versions, and expected  
595       versus observed data. A record is similar to a row in a SQL view. It is a projection of properties. The same  
596       property may appear in multiple records for the same item or relationship. The record may have no  
597       properties, in which case it serves as a marker.

598       Each record may have the following metadata properties that describe the record itself (as opposed to  
599       properties that describe the item or relationship):

600       • an ID that is unique within the scope of its associated item or relationship and that serves as a  
601       key (optional if there is only one record for the item or relationship)

602       • the date/time the record was last modified (optional)

603       • a baseline ID that may be used to indicate the expected (authorized) configuration baseline this  
604       record represents (optional)

605       • a snapshot ID that may be used to indicate the configuration observations this record  
606       represents (optional)

607       Each record has exactly one "record type". Note that a record type may extend one or more other record  
608       types, as described in 8.2.2.3. A record type is:

609       • A characterization of an item or relationship.

610       • A collection of properties that can be used to describe an item or relationship. The properties  
611       may be simple or complex XML elements.

612       • A record type may be used in a query to limit the items or relationships returned by a query  
613       operation to instances with a record considered by the query service to be of the requested  
614       type.

615       A record type may also be the QName of the first child of a record element in a query response.

## 616    5.5.2   Common Data Element Types

617       The `cmdbf:MdrScopedIdType` is used in several places to identify an item or relationship. It is described  
618       here for convenience so other sections of this document may refer to it without repeating the definition.

619       The `<instanceId>` element is of the type of `cmdbf:MdrScopedIdType`. The pseudo-schema of the  
620       `<instanceId>` element is as follows:

```
621 <instanceId>
622   <mdrId>xs:anyURI</mdrId>
623   <localId>xs:anyURI</localId>
624 </instanceId>
```

625       This can be abbreviated in a pseudo schema as the following:

```
626 <instanceId>cmdbf:MdrScopedIdType</instanceId>
```

627       The `cmdbf:MdrScopedIdType` is composed of a pair of URIs. The first URI, `<mdrId>`, is the ID of the  
628       MDR that assigned this instance ID to the instance. The second URI, `<localId>`, is the ID that uniquely  
629       identifies the instance within the MDR. The combination of these two URIs identifies the instance in a  
630       globally unique way. There is no expectation that these two URIs are able to be de-referenced.

631 Every `<record>` element has exactly one child element of unrestricted content (which is typically used to  
632 describe the item or relationship with which the record is associated), followed by an optional (if there is  
633 only one record associated with the item or relationship) `<recordMetadata>` element that contains  
634 common information about the record itself.

635 The `<recordMetadata>` element may contain these properties:

- 636 • `recordId`: the unique ID of the record in the MDR. If there is more than one record for an item or  
637 a relationship, the `recordId` is required.
- 638 • `lastModified`: the time/date the record was last modified in ISO 8601 format. The applicable time  
639 zone or UTC shall be indicated.
- 640 • `baselineId`: the name or other identifier used to group records into a particular baseline  
641 configuration. A value of "0" indicates that this record is not part of any baseline configuration.
- 642 • `snapshotId`: the name or other identifier used to group records observed in a configuration  
643 snapshot (discovery). A value of "0" indicates that this record is not part of any snapshot  
644 configuration.
- 645 • `extensibility elements`: additional metadata elements not defined by the specification may also  
646 be included

## 647 6 Query Service

### 648 6.1 Overview

649 The Query Service can be provided by MDRs and federating CMDBs (see Table 1 – Service Usage  
650 Patterns on page 18). It provides a way to access the items and relationships that the provider (MDR or  
651 federating CMDB) has access to, whether this provider actually holds the data or federates the source of  
652 the data. The Query Service contains a `GraphQL` operation that can be used for anything from a  
653 simple instance query to a much more complex topological query.

654 A `GraphQL` request describes the items and relationships of interest in the form of a graph.  
655 Constraints can be applied to the nodes (items) and edges (relationships) in that graph to further refine  
656 them. The `GraphQL` response contains the items and relationships that, through their combination,  
657 compose a graph that satisfies the constraints of the graph in the query.

658 The subsequent subclauses provide a more complete description of the request and response messages  
659 for the `GraphQL` operation. Examples are provided in ANNEX D.

### 660 6.2 GraphQL Operation Outline

661 A `GraphQL` request consists of a `<query>` element that contains `<itemTemplate>` and  
662 `<relationshipTemplate>` elements. Content selectors and constraints can be used inside  
663 `<itemTemplate>` or `<relationshipTemplate>` elements, and have the same form in both.

664 In addition to constraints, `<relationshipTemplate>` elements also contain a `<sourceTemplate>`  
665 and a `<targetTemplate>` element. These elements each point (using the `xs:ID/xs:IDREF` mechanism)  
666 to an `<itemTemplate>`.

667 The pseudo-schema for the payload of a `GraphQL` request is as follows:

```
668 <query>  
669   <itemTemplate id="xs:ID" suppressFromResult="xs:boolean" ?>  
670     (<contentSelector ...>...</contentSelector> ?  
671     <instanceIdConstraint>...</instanceIdConstraint> ?
```

```

672 <recordConstraint>
673   <recordType ... /> *
674   <propertyValue ...>...</propertyValue> *
675   <xpathConstraint>...</xpathConstraint> ?
676 </recordConstraint> *)
677   xs:any
678 </itemTemplate> *
679 <relationshipTemplate id="xs:ID" suppressFromResult="xs:boolean" ?>
680   (<contentSelector ...>...</contentSelector> ?
681   <instanceIdConstraint>...</instanceIdConstraint> ?
682   <recordConstraint>
683     <recordType>...</recordType> *
684     <propertyValue>...</propertyValue> *
685     <xpathConstraint>...</xpathConstraint> ?
686   </recordConstraint> *)
687   <sourceTemplate ref="xs:IDREF" minimum="xs:int"?
688     maximum="xs:int"?/> ?
689   <targetTemplate ref="xs:IDREF" minimum="xs:int"?
690     maximum="xs:int"?/> ?
691   <depthLimit ... /> ?
692   xs:any
693 </relationshipTemplate> *
694 </query>

```

695 The syntax and semantics for each constraint element are provided in later clauses (for  
696 <instanceIdConstraint> see 6.4.1, for <propertyValue> see 6.4.2.2, for <recordType> see  
697 6.4.2.1, and for <xpathConstraint> see 6.4.2.3). The evaluation of a constraint on an item or  
698 relationship returns a Boolean expression. If the value of the Boolean expression is true, then the item or  
699 relationship is deemed to satisfy the defined constraint.

700 Templates are used to identify matching items and relationships to be returned in the graph response.

701 The optional “suppressFromResult” attribute, if present and set to true, indicates that the items or  
702 relationships that correspond to the template carrying the attribute should be suppressed from the result.  
703 Templates with this attribute set to true are still meaningful in that it may help constrain other templates in  
704 the query. For example, in order to retrieve all items that have a “dependsOn” relationship with application  
705 “foo”, the query may set this attribute to true on the template for the “foo” item and the template for the  
706 “dependsOn” relationship but not on the template for the items on which “foo” depends. Only the latter  
707 items would appear in the response. If the “suppressFromResult” attribute is not present or set to false on  
708 a template, then all the selected instances for this template are returned in the query result.

### 709 6.2.1 itemTemplate

710 An item matches an <itemTemplate> if and only if all of the following provisions are true:

- 711 • The item satisfies all the constraints defined by the <itemTemplate>. (In effect, an implicit  
712 AND joins the constraints.)
- 713 • For every <relationshipTemplate> that points to the <itemTemplate> as its  
714 sourceTemplate, there is a relationship matching this <relationshipTemplate> that has the  
715 item as its source.

- 716       • For every <relationshipTemplate> that points to the <itemTemplate> as its  
717       targetTemplate, there is a relationship matching this <relationshipTemplate> that has the  
718       item as its target.

719 An item can match more than one <itemTemplate> inside a given query. When this is the case, the  
720 item appears in the response once for each matching <itemTemplate> (unless suppressed by the  
721 "suppressFromResult" attribute).

722 An item template will not return relationship instances.

## 723 6.2.2 relationshipTemplate

724 A relationship matches a <relationshipTemplate> if and only if all of the following provisions are  
725 true:

- 726       • The relationship meets all the constraints in the <relationshipTemplate>. (In effect, an  
727       implicit AND joins the constraints.)
- 728       • The source item of the relationship matches the <itemTemplate> referenced as  
729       <sourceTemplate> by the <relationshipTemplate>.
- 730       • The target item of the relationship matches the <itemTemplate> referenced as  
731       <targetTemplate> by the <relationshipTemplate>.
- 732       • The cardinality conditions on the <sourceTemplate> and <targetTemplate> elements are  
733       satisfied, as defined by the @minimum and @maximum attributes defined 6.2.2.1.
- 734       • The depth, or the number of edges between source and target nodes in the graph, satisfies the  
735       <depthLimit> condition defined in 6.2.2.2.

736 Items, which do not have a source or target, cannot match a <relationshipTemplate>.

### 737 6.2.2.1 relationshipTemplate/sourceTemplate and relationshipTemplate/targetTemplate

738 The <sourceTemplate> and <targetTemplate> elements each refer to an <itemTemplate>  
739 element using the required @ref attribute. The value of the @ref attribute shall match the value of the @id  
740 attribute of an <itemTemplate> element in the query.

741 Additionally, <sourceTemplate> and <targetTemplate> elements may have the following optional  
742 attributes:

743       **@minimum** – If n is the value of the @minimum attribute, there shall be at least n relationships  
744       matching the <relationshipTemplate> that share the same source or target item. For example,  
745       a query to find computers that at least five services depend upon might specify minimum="5" on a  
746       <sourceTemplate> that selects services, combined with a <targetTemplate> that selects  
747       computers and other constraints that select a 'dependsOn' relationship.

748       **@maximum** – If n is the value of the @maximum attribute, there may be at most n relationships  
749       matching the <relationshipTemplate> that share the same source or target item.

### 750 6.2.2.2 relationshipTemplate/depthLimit

751 The <depthLimit> element is used to extend the relationship template to traverse multiple edges and  
752 nodes. For example, this element may be used to find all the components of an aggregate system, or all  
753 the dependencies of a business service, even if these items are not directly related to the item in  
754 question. This extended relationship is also called a "relationship chain."



755 The pseudo-schema of the <depthLimit> element is as follows:

```
756 <depthLimit maxIntermediateItems="xs:positiveInteger" ?
757   intermediateItemTemplate="xs:IDREF" />
```

758 **@maxIntermediateItems** – The maximum number of intermediate items in the relationship chain  
 759 between source and target items. A value of 1 indicates that the <relationshipTemplate> can  
 760 traverse one intermediate item between the source item and target item. This attribute is optional. If it  
 761 is not present, then the number of intermediate items between the source and the target is unlimited.

762 **@intermediateItemTemplate** – The value of the intermediateItemTemplate corresponds to the @id  
 763 attribute of an <itemTemplate> element that is used as a prototype for intermediate items in the  
 764 relationship chain. The value of the @intermediateItemTemplate attribute is also used to represent  
 765 the intermediate items in the <nodes> element of the query response.

## 766 6.3 Content Selection

767 The <contentSelector> element determines how instances matching the template are returned in the  
 768 response. If a template does not contain a <contentSelector> element, all matching instances and  
 769 associated records are returned in the response. The term "instance" means either an item or a  
 770 relationship.

771 If a template contains a <contentSelector> element, the records and properties returned for the  
 772 instances that match this template are limited to those explicitly selected. Records and properties are  
 773 explicitly selected by specifying their namespace and local name in the <selectedRecordType>  
 774 element or an XPath expression in the <xpathSelector> element. The use of  
 775 <selectedRecordType> and <xpathSelector> are mutually exclusive per content selector.

776 The pseudo-schema of the <contentSelector> element is as follows:

```
777 <contentSelector>
778   (<selectedRecordType namespace="xs:anyURI" localName="xs:NCName" >
779     <selectedProperty namespace="xs:anyURI" localName="xs:NCName" /> *
780   </selectedRecordType> * |
781   <xpathSelector dialect="xs:anyURI">
782     <prefixMapping prefix="xs:NCName" namespace="xs:anyURI"/> *
783     <expression>xs:string</expression>
784   </xpathSelector> ?)
785 </contentSelector>
```

### 786 6.3.1 contentSelector

787 The use of the <contentSelector> element affects the contents of the matching instances in the  
 788 response as follows:

- 789 • <contentSelector /> (empty element)

790 The instances matching this template are returned with no record content in the response. This may  
 791 be useful if all that is required is the instanceId of instances matching this template.

#### 792 6.3.1.1 contentSelector/selectedRecordType

793 If <selectedRecordType> is used without any <selectedProperty> child elements, all properties  
 794 (child elements) of all records of the selected type are returned in the response.

795 At the discretion of the query service, the response may contain a record type that is an extension (as  
 796 described in 8.2.2.3) of the selected record type. For example, the following query limits the response to

797 records with a record type with namespace="http://example.com/models" and  
798 localName="Computer".

```
799 <query>
800   <itemTemplate id="computers">
801     <contentSelector>
802       <selectedRecordType namespace="http://example.com/models"
803         localName="Computer">
804     </selectedRecordType>
805   </contentSelector>
806 </itemTemplate>
807 </query>
```

808 A valid response to this query could contain records with a record type of  
809 namespace="http://example.com/models" and localName="LinuxComputer", as long as the  
810 record type with localName="LinuxComputer" is defined as an extension of the record type with  
811 localName="Computer" using the mechanism described in 8.2.2.3.

#### 812 6.3.1.1.1 contentSelector/selectedRecordType/selectedProperty

813 If <selectedProperty> elements are included in a <selectedRecordType> element, only the  
814 selected properties of the selected record types are returned in the response.

815 EXAMPLE: In the following example, only the "name" and "telephone" properties in the  
816 <http://example.com/models/people> namespace get returned for the items that match the "user" <itemTemplate>.

```
817 <query>
818   <itemTemplate id="user">
819     <contentSelector>
820       <selectedRecordType namespace="http://example.com/models"
821         localName="people">
822         <selectedProperty namespace="http://example.com/models/people"
823           localName="name"/>
824         <selectedProperty namespace="http://example.com/models/people"
825           localName="telephone"/>
826       </selectedRecordType>
827     </contentSelector>
828     ...
829   </itemTemplate>
830 </query>
```

831 Whether or not individual properties are selected, the contents of an item or relationship in the response  
832 are always in the form of <record> elements, as follows, or in a <propertySet> element, which is  
833 described in 6.6.1:

```
834 <record>
835   <recordTypeQName>
836     <propertyQName>xs:any</propertyQName> *
837   </recordTypeQName>
838   <recordMetadata>
839     <recordId>xs:any</recordId>
840     ...
841   </recordMetadata>
842 </record> *
```

843 A record type may extend multiple record types, as shown in the example on the right hand side of  
 844 Figure 6 in 8.2.2.3. For each record of an item, regardless of how many record types may describe a  
 845 subset of the record properties and regardless of how many  
 846 <contentSelector>/<selectedRecordType> elements select all or part of this record, the query  
 847 response shall contain at most one record or property set (see 6.6.1 for a description of a property set).  
 848 The record type of the returned record or property set shall be a record type that contains all the  
 849 properties to be returned. Using the same example on the right hand side of Figure 6, a query that selects  
 850 the faxNumber property of FaxMachine could be satisfied by returning either a FaxMachine or  
 851 MultiFunctionPrinter record or property set.

### 852 6.3.1.2 contentSelector/xpathSelector

853 The use of the <xpathSelector> element may be used to selects parts of complex models or for  
 854 complex selection criteria. For example, an item template has matched an item with the following record:

```
855 <record>
856   <ex:ComputerSystem xmlns:ex="http://www.example.org/cs">
857     ...
858     <ex:NetworkInterfaces>
859       <ex:ip>1.2.3.4</ex:ip>
860       <ex:ip>2.3.4.5</ex:ip>
861     </ex:NetworkInterfaces>
862     ...
863   </ex:ComputerSystem>
864   ...
865 </record>
```

866 If the <xpathSelector> is as follows:

```
867 <xpathSelector
868   dialect="http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1">
869   <prefixMapping prefix="ex" namespace="http://www.example.org/cs" />
870   <expression>
871     /ex:ComputerSystem/ex:NetworkInterfaces/ex:ip
872   </expression>
873 </xpathSelector>
```

874 The record returned would be:

```
875 <record>
876   <ex:ip>1.2.3.4</ex:ip>
877   <ex:ip>2.3.4.5</ex:ip>
878 </record>
```

#### 879 6.3.1.2.1 contentSelector/xpathSelector/@dialect

880 The dialect corresponds to a particular version or profile of XPath represented by the URI value. See 6.5  
 881 for more information on XPath dialects.

#### 882 6.3.1.2.2 contentSelector/xpathSelector/prefixMapping

883 Each <prefixMapping> child element of the <xpathConstraint> element defines a namespace  
 884 declaration for the XPath evaluation. The prefix for this declaration is provided by the  
 885 <prefixMapping>/@prefix attribute and the namespace URI is provided by the  
 886 <prefixMapping>/@namespace attribute. These prefix-namespace pairings shall be added to the  
 887 namespace declarations of the XPath processor.

### 888 6.3.1.2.3 contentSelector/xpathSelector/expression

889 The <expression> element contains an XPath expression to be evaluated according to the chosen  
890 dialect against each <record> element contained in an item or relationship that has satisfied all of the  
891 constraints. The evaluation result is then transformed and normalized into a single DOM node according  
892 to the mechanism prescribed by the dialect. See 6.5 for more information on XPath normalization.

893 If that response DOM node has any children, then the record is selected and those children are appended  
894 to the <record> element.

## 895 6.4 Constraints

896 Constraints are used to restrict the instances returned based on properties of the instances and  
897 associated records.

### 898 6.4.1 instanceIdConstraint

899 The <instanceIdConstraint> element is used to point to specific instances by instance ID. The  
900 pseudo-schema of this element is as follows:

```
901 <instanceIdConstraint>  
902   <instanceId>cmdbf:MdrScopedIdType</instanceId> +  
903 </instanceIdConstraint>
```

904 There can be at most one <instanceIdConstraint> in an <itemTemplate> or a  
905 <relationshipTemplate> element.

906 More than one instance ID may be attached to one instance. For example, a federating CMDB may know,  
907 for a given reconciled instance, instance IDs provided by each of the MDRs that have content about the  
908 instance, plus possibly an additional instance ID for the instance assigned by the federating CMDB itself.

909 The constraint is satisfied if one of the known instance IDs for the instance matches one of the requested  
910 values (that is, if both the <mdrId> and the <localId> match using string comparison).

### 911 6.4.2 recordConstraint

912 The <recordConstraint> element is used to point to specific record types and related properties to be  
913 evaluated.

914 The pseudo-schema of this element is as follows:

```
915 <recordConstraint>  
916   <recordType namespace="xs:anyURI"  
917     localName="xs:NCName" /> *  
918   <propertyValue> ... </propertyValue> *  
919   <xpathConstraint> ... </xpathConstraint> ?  
920   xs:any  
921 </recordConstraint>
```

922 The <recordConstraint> element can appear any number of times inside an <itemTemplate> or a  
923 <relationshipTemplate>.

#### 924 6.4.2.1 recordConstraint/recordType

925 The <recordType> element can appear any number of times inside a <recordConstraint>  
926 element.

927 One way for this constraint to be satisfied is if the instance has a record of that type. More specifically, if  
 928 the instance contains a record element that has, as the first child element, an element in the namespace  
 929 corresponding to the value of the <recordType>/@namespace attribute and where the local name of  
 930 that first child element is the value of the <recordType>/@localName attribute. The constraint could  
 931 also be satisfied by an instance with a record that is an extension of that QName, as described in 8.2.2.3.  
 932 (For example, comp:Linux might be defined as an extension of comp:OperatingSystem.)

#### 933 6.4.2.2 recordConstraint/propertyValue

934 Each instance is associated with zero or more records. These records contain properties whose values  
 935 are accessible through an XML representation of the instance. The <propertyValue> element can only  
 936 be used on properties that have a type that is a subtype of the xs:anySimpleType type. While the type  
 937 must be known, it is not required that an XML schema definition of the property be available.

938 The <propertyValue> element is not applicable to properties that are defined as a complex type.

939 The pseudo-schema of this element is as follows:

```

940 <propertyValue namespace="xs:anyURI"
941     localName="xs:NCName"
942     recordMetadata="xs:boolean" ?
943     matchAny="xs:boolean" ? >
944     <equal caseSensitive="xs:boolean"? negate="xs:boolean"? >
945         xs:anySimpleType
946     </equal> *
947     <less negate="xs:boolean"? >xs:anySimpleType</less> ?
948     <lessOrEqual negate="xs:boolean"? >xs:anySimpleType</lessOrEqual> ?
949     <greater negate="xs:boolean"? >xs:anySimpleType</greater> ?
950     <greaterOrEqual negate="xs:boolean"? >
951         xs:anySimpleType
952     </greaterOrEqual> ?
953     <contains caseSensitive="xs:boolean"? negate="xs:boolean"? >
954         xs:string
955     </contains> *
956     <like caseSensitive="xs:boolean"? negate="xs:boolean"? >
957         xs:string
958     </like> *
959     <isNull negate="xs:boolean"? /> ?
960     xs:any
961 </propertyValue>
  
```

962 The <propertyValue> element can appear any number of times in <recordConstraint>. Its  
 963 namespace and localName attributes define the QName of the property being tested. If there are one or  
 964 more <recordType> elements in the enclosing <recordConstraint>, they define the record types  
 965 against which to evaluate the constraint. If there are no <recordType> elements, the  
 966 <propertyValue> element is evaluated against all record types.

967 The child elements of <propertyValue> are called operators. A <propertyValue> constraint is  
 968 considered to be satisfied if the operators return a positive (true) result for one or more records  
 969 associated with the instance (logical OR across the records).

970 The operators are largely defined in terms of [XPath 2.0](#) comparison operators. This does not require that  
 971 an [XPath 2.0](#) implementation be used but only that the operators be evaluated in a way that is consistent  
 972 with the [XPath 2.0](#) definitions, as described in 6.4.2.3.

973        **@recordMetadata** – The value of this attribute indicates that the property to be evaluated is in the  
974        <recordMetadata> element of the record.

975        **@matchAny** – The value of this attribute defines whether the operators inside that element are  
976        logically AND-ed or OR-ed. The default value for the matchAny attribute is false. If the value of the  
977        matchAny attribute is false, the constraint returns a positive result for an instance if the instance has  
978        a record that contains the property identified by the QName and if the value of that property satisfies  
979        *all* the operators in the constraint (logical AND). If the value of the matchAny attribute is true, the  
980        constraint returns a positive result for an instance if the instance has a record that contains the  
981        property identified by the QName and if the value of that property satisfies *at least one* of the  
982        operators in the constraint (logical OR).

#### 983    **6.4.2.2.1 recordConstraint/propertyValue/equal**

984    This operator is defined in terms of the [XPath 2.0](#) value comparison operator "eq". To evaluate, the  
985    operand on the left is the property value from the record and the operand on the right is the value of the  
986    constraint from the query. The type of the value of the constraint shall be interpreted to be of the same  
987    type as the value from the property in the record. This operator is valid for properties of any simple type.  
988    A list of comparison behaviors is available in [XPath 2.0](#), "Appendix B.2 – Operator Mappings".

#### 989    **6.4.2.2.2 recordConstraint/propertyValue/less, 990            recordConstraint/propertyValue/lessOrEqual, 991            recordConstraint/propertyValue/greater, and 992            recordConstraint/propertyValue/greaterOrEqual**

993    These operators are defined in terms of the [XPath 2.0](#) value comparison operators "lt", "le", "gt", and "ge",  
994    respectively. To evaluate, the operand on the left is the property value from the record and the operand  
995    on the right is the value of the constraint from the query. The type of the value of the constraint shall be  
996    interpreted to be of the same type as the value from the property in the record. These operators are valid  
997    only for properties that are numerals, dates, and strings. A list of comparison behaviors is available in  
998    [XPath 2.0](#), "Appendix B.2 – Operator Mappings". For example, if a property is of type date, the operator  
999    <less>2000-01-01T00:00:00</less> returns true if the property value is a date before the year  
1000    2000. If the property value is a string, then "2000-01-01T00:00:00" is interpreted as a string and  
1001    compared with the property value using string comparison.

#### 1002    **6.4.2.2.3 recordConstraint/propertyValue/contains**

1003    This operator is mapped to the [XPath 2.0](#) function fn:contains(). It is valid only for properties of type string  
1004    and used to test whether the property value contains the specified string as a substring. The result of the  
1005    contains operator is as if the fn:contains() function were executed with the first parameter being the  
1006    property value and the second parameter being the string specified.

#### 1007    **6.4.2.2.4 recordConstraint/propertyValue/like**

1008    This operator is similar in functionality to the SQL LIKE clause. The operator works like the equal operator  
1009    with the inclusion of the following two special characters:

- 1010        • The underscore character ( "\_" ) acts as a wild card for any single character.
- 1011        • The percent sign ( "%" ) acts as a wild card for zero or more characters.

1012    To escape the wild cards, the backslash ( "\ " ) can be used. For example,  
1013    <like>Joe\\_Smith%</like> tests whether the property value starts with the string "Joe\_Smith" and  
1014    would match values such as "Joe\_Smith", "Joe\_Smith123", and "Joe\_Smith\_JR". It would not match  
1015    "JoeHSmith123". A double backslash ("\\") represents the single backslash string ("\").

1016 **6.4.2.2.5 recordConstraint/propertyValue/isNull**

1017 This operator tests whether the element corresponding to the property is "nilled". It is equivalent to the  
 1018 result of applying the [XPath 2.0](#) "fn:nilled" function on the element corresponding to the property.

1019 **6.4.2.2.6 Additional Attributes**

1020 The following additional attributes are defined for operator elements:

1021 **@caseSensitive** – This is an optional attribute for the equal, contains, and like operators. The  
 1022 default value is true. If the property value of the record is an instance of xs:string and the  
 1023 caseSensitive attribute is false, the string comparison is case-insensitive. More precisely, the result  
 1024 of the comparison is as if the [XPath 2.0](#) function fn:upper-case() was called on both the property  
 1025 value and the string value before comparison. If the property value of the record is not an instance of  
 1026 an xs:string, the caseSensitive attribute has no impact on the comparison.

1027 **@negate** – This is an optional attribute for all operators. The default value is false. When the negate  
 1028 attribute is true, the result of the comparison is negated.

1029 Table 2 summarizes which operators are supported for the various XSD built-in datatypes. Unless  
 1030 explicitly specified, the caseSensitive attribute is not supported.

1031 **Table 2 – Operators Supported for XSD Built-in Datatypes**

Built-in Datatypes	equal	isNull	less, lessOrEqual, greater, greaterOrEqual	contains	like
"String-related types" (String, anyURI, and types derived from string)	Yes, including the optional caseSensitive attribute	Yes	Yes	Yes, including the optional caseSensitive attribute	Yes, including the optional caseSensitive attribute
"Time-related and numeric types" (duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth, float, double, decimals, and all types derived from decimals)	Yes	Yes	Yes	No	No
"Others" (Boolean, QName, NOTATION, base64Binary, and hexBinary)	Yes	Yes	No	No	No

1032 If more than one property uses the same QName, the comparison has to hold true for only one of the  
 1033 property values.

1034 **EXAMPLE 1:** Consider the following example for a computer with three IP addresses:

```

1035 <comp:ComputerConfig xmlns:comp="http://example.com/computers">
1036   ...
1037   <comp:ip>1.2.3.4</comp:ip>
1038   <comp:ip>1.2.3.5</comp:ip>
1039   <comp:ip>1.2.3.6</comp:ip>
1040   ...
1041 </comp:ComputerConfig>
    
```

1042 The following property constraint would return a positive result:

```
1043 <recordConstraint>
1044   <propertyValue namespace="http://example.com/computers"
1045     localName="ip">
1046     <equal>1.2.3.5</equal>
1047   </propertyValue>
1048 </recordConstraint>
```

1049 When the negate attribute is used on a list of properties, the negation is taken after the operator  
1050 executes. When negating the equal operator, a positive result is returned when none of the properties are  
1051 equal to the given value.

1052 EXAMPLE 2: For example, on the same computer with three IP addresses:

```
1053 <recordConstraint>
1054   <propertyValue namespace="http://example.com/computers"
1055     localName="ip">
1056     <equal negate="true">1.2.3.5</equal>
1057   </propertyValue>
1058 </recordConstraint>
```

1059 The property constraint would remove the item above from the result set because the equality comparison matches  
1060 one IP address in the list.

1061 Similarly, `<less negate="true">12</less>` is equivalent to  
1062 `<greaterOrEqual>12</greaterOrEqual>` if there is only one instance of the property being tested.  
1063 But if there is more than one instance of the property, then the first operator is true if all of the instances  
1064 have a value of more than 12, while the second one is true if at least one of the instances has a value of  
1065 more than 12.

1066 EXAMPLE 3: The following is a simple example of using `<propertyValue>`. "Manufacturer" is a property defined  
1067 in the "http://example.com/Computer" namespace. The constraint is testing whether the instance has a record  
1068 containing this property and where the value of the property is "HP".

```
1069 <recordConstraint>
1070   <propertyValue namespace="http://example.com/Computer"
1071     localName="Manufacturer" >
1072     <equal>HP</equal>
1073   </propertyValue>
1074 </recordConstraint>
```

1075 EXAMPLE 4: The following is a more complex example. The `<itemTemplate>` matches any item that has a  
1076 CPUCount greater than or equal to 2, for which the OSName property contains "Linux" (with that exact mix of upper  
1077 and lower case letters), and for which the OSName property also contains either "ubuntu" or "debian" (irrespective of  
1078 case).

```
1079 <itemTemplate id="linuxMachine">
1080   <recordConstraint>
1081     <propertyValue namespace="http://example.com/computers"
1082       localName="CPUCount">
1083       <greaterOrEqual>2</greaterOrEqual>
1084     </propertyValue>
1085     <propertyValue namespace="http://example.com/computers"
1086       localName="OSName">
1087       <contains>Linux</contains>
1088     </propertyValue>
1089     <propertyValue namespace="http://example.com/computers"
```



```

1090         localName="OSName"
1091         matchAny="true">
1092     <contains caseSensitive="false">ubuntu</contains>
1093     <contains caseSensitive="false">debian</contains>
1094 </propertyValue>
1095 <recordConstraint/>
1096 </itemTemplate>

```

### 1097 6.4.2.3 recordConstraint/xpathConstraint

1098 The <xpathConstraint> element provides an alternate mechanism to constrain items and  
1099 relationships. The pseudo-schema of this element is as follows:

```

1100 <xpathConstraint dialect="xs:anyURI">
1101     <prefixMapping prefix="xs:NCName" namespace="xs:anyURI" /> *
1102     <expression>xs:string</expression>
1103 </xpathConstraint>

```

1104 The <xpathConstraint> element may appear once inside a <recordConstraint> inside an  
1105 <itemTemplate> or <relationshipTemplate> element. It can only be used in conjunction with a  
1106 <propertyValue> constraint if the <propertyValue> constraint in question applies to record  
1107 metadata. In other words, if a <recordConstraint> contains a <xpathConstraint> then it can only  
1108 contain <propertyValue> elements, which have the recordMetadata attribute set to true. When  
1109 such metadata-related <propertyValue> elements are used together with a <xpathConstraint>  
1110 element, they are all ANDed together: to be selected, an item or relationship shall have a record for which  
1111 the metadata meets all the constraints in the <propertyValue> elements and the record content  
1112 satisfies the XPath constraint.

#### 1113 6.4.2.3.1 recordConstraint/xpathConstraint/@dialect

1114 The dialect corresponds to a particular version or profile of XPath represented by the URI value. See 6.5  
1115 for more information on XPath dialects.

#### 1116 6.4.2.3.2 recordConstraint/xpathConstraint /prefixMapping

1117 Each <prefixMapping> child element of the <xpathConstraint> element defines a namespace  
1118 declaration for the XPath evaluation. The prefix for this declaration is provided by the  
1119 <prefixMapping>/@prefix attribute and the namespace URI is provided by the  
1120 <prefixMapping>/@namespace attribute. These prefix-namespace pairings shall be added to the  
1121 namespace declarations of the XPath processor.

#### 1122 6.4.2.3.3 recordConstraint/xpathConstraint/expression

1123 The <expression> element contains an XPath expression to be evaluated according to the specified  
1124 dialect.

1125 The <xpathConstraint> is satisfied if the evaluation result's boolean value is true. The boolean value  
1126 of the evaluation result is the same result as running the XPath 1 function boolean() on the results of a  
1127 XPath 1 evaluation or the XPath 2 function fn:boolean() on the results of a XPath 2 evaluation.

1128 EXAMPLE: In the following example, "name" is a property defined in the "http://example.com/people" namespace.  
1129 The constraint tests whether the instance has a record containing this property where the value of the  
1130 property is "Pete the Lab Tech". In this example, no metadata is selected by the expression.

```

1131 <itemTemplate>
1132     <recordConstraint>
1133         <xpathConstraint

```

```

1134     dialect=" http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1">
1135     <prefixMapping prefix="hr" value="http://example.com/people"/>
1136     <expression>/hr:ContactInfo[hr:name = "Pete the Lab Tech"]
1137     </expression>
1138     </xpathConstraint>
1139     </recordConstraint>
1140 </itemTemplate>

```

## 1141 6.5 XPath Expressions and Normalization

1142 XPath may be used as a more flexible way to constrain what items/relationships are matched in a query  
 1143 and/or to select the record content returned for selected items/relationships. When used as a selector and  
 1144 a constraint, the client and server need to have a common understanding of how they will interpret and  
 1145 process the XPath expression. This is done through specifying an XPath dialects and a corresponding  
 1146 URI. This specification defines two dialects that may be used as either a selector or as a constraint:

- 1147 • "<http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1>" indicates that the expression  
 1148 corresponds to an XPath 1.0 expression.
- 1149 • "<http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath2>" indicates that the expression  
 1150 corresponds to an XPath 2.0 expression.

1151 Other dialects may be defined in future versions of this specification or in other specifications.

1152 Implementations are free to provide its own URI for a dialect that is not defined in the specification.

1153 To enable serialization and to simplify the processing of the XPath selector, the XPath selector evaluation  
 1154 result is run through a transformation and then a normalization process. The transformation process  
 1155 transforms attribute nodes into element nodes; this allows them to be serialized later on. Next, this result  
 1156 is run through the normalization process which creates a single DOM node with the selection result nodes  
 1157 as children.

1158 The normalization process shall throw a cmdbf:XPathSerializationFault fault if there is unsupported  
 1159 serialization input from the transformation process. For the XPath 1.0 normalization process, the  
 1160 serialization input shall either be a simple value or a nodeset made up of only element nodes. For the  
 1161 XPath 2.0 normalization process, the serialization input shall not contain any namespace, comment, or  
 1162 processing instruction nodes.

### 1163 6.5.1 XPath 1.0 Dialect

1164 This dialect indicated by the URI of <http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1> is specified for  
 1165 XPath 1.0 support, subject to the conditions described in 6.5.3 and 6.5.4.

1166 The XPath expression is evaluated in the following context:

Component	Value
Context Node	The first child of the <record> element
Context Position	1
Context Size	1
Variable Binding	None
Function Libraries	Core function library
Namespace Declarations	Prefixes bound via <prefixMapping> element

1167 **6.5.2 XPath 2.0 Dialect**

1168 This dialect indicated by the URI of <http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath2> is specified for  
1169 XPath 2.0 support, subject to the conditions described in 6.5.3 and 6.5.5.

1170 The XPath expression is evaluated in the following context:

Component	Value
XPath 1.0 Compatibility Mode	False
Statically known namespaces	Prefixes bound via <prefixMapping> element
Default element/type namespace	None
Default function namespace	None
In-scope variables	None
Context item static type	element([namespace of this specification], record)
Function signatures	Functions defined in <a href="#">XQuery 1.0 and XPath 2.0 Functions and Operators</a>
Context item	The first child of the <record> element
Context position	1
Context size	1
Current date and time	Time on server when request was made

1171 **6.5.3 XPath Selector Transformation**

1172 The transformation allows for selecting XML attributes. This is done through mapping an XML attribute to  
1173 a <attributeNode> element:

- 1174 • The XML attribute value is mapped to the @value of the <attributeNode>.
- 1175 • The XML attribute local name is mapped to the @localName of the <attributeNode>.
- 1176 • The XML attribute namespace is mapped to the @namespace of the <attributeNode>.

1177 The pseudo schem of <attributeNode> looks like:

```
1178 <cmdbf:attributeNode namespace="xs:anyUri"
1179   localName="xs:NCName" value="xs:anySimpleType" />
```

1180 The result is as if the following XSLT template was matched to the selection result:

```
1181 <xsl:template match="@*">
1182   <cmdbf:attributeNode>
1183     <xsl:attribute name="namespace">
1184       <xsl:value-of select="namespace-uri(.)" /></xsl:attribute>
1185     <xsl:attribute name="localName">
1186       <xsl:value-of select="local-name(.)" /></xsl:attribute>
1187     <xsl:attribute name="value">
1188       <xsl:value-of select="." /></xsl:attribute>
1189   </cmdbf:attributeNode>
1190 </xsl:template>
```

1191 The "xsl" prefix is bound to XSL 1.0 or 2.0 depending on whether an XPath 1 or XPath 2 evaluation result  
1192 was input.

1193 Here's an example of how an attribute would be mapped. If the record is:

```
1194 <hr:ContactInfo xmlns:hr="http://example.com/hr" changeby="jsmith">
1195 ...
1196 </hr:ContactInfo> <cmdbf:attributeNode>
```

1197 The result of the content selector with an XPath selector with the expression "hr:ContactInfo/@changeby"  
1198 would be:

```
1199 <cmdbf:attributeNode namespace=""
1200     localName="changeby"
1201     value="jsmith" />
```

## 1202 6.5.4 XPath 1.0 Normalization

1203 The selection evaluation result set for XPath 1.0 is then normalized:

1204 Create a new sequence S.

1205 If the result set is empty, then add a zero length string to the sequence S. If the result set contains a  
1206 string, a number, or a boolean, run the XPath string() on the item to get the string value and add this  
1207 string value to the sequence S. If the result set is a node set and contains any node other than a element  
1208 node, throw a cmdbf:XPathSerializationFault; if the result is a node set and only contains nodes of type  
1209 element, then add these nodes to the sequence S.

1210 Create a new DocumentFragment named DF. For each item in S, if the item is a string, create a text node  
1211 and add the text node to DF. Or, if the item is an element node, add the element node to DF.

1212 The result of this normalization process is a DocumentFragment named DF.

## 1213 6.5.5 XPath 2.0 Normalization

1214 The selection result set for XPath 2.0 results is then normalized as defined in Section 2 "Sequence  
1215 Normalization" of the [XSLT 2.0 and XQuery 1.0 Serialization](#) specification. If the serialization input  
1216 contains any namespace, comment, or processing instruction nodes, or any other serialization error  
1217 occurs, cmdbf:XPathSerializationFault shall be thrown. The serialization error definition is from  
1218 <http://www.w3.org/TR/xslt-xquery-serialization/#serial-err>.

## 1219 6.6 GraphQuery Response

1220 The pseudo-schema for the GraphQuery response message is as follows:

```
1221 <queryResult>
1222   <nodes templateId="xs:ID">
1223     <item>
1224       <record>
1225         xs:any
1226         |
1227         <propertySet namespace="xs:anyURI" localName="xs:NCName" >
1228           xs:any *
1229         </propertySet>
1230       <recordMetadata>
1231         <recordId>...</recordId> ?
1232         <lastModified>...</lastModified> ?
```

```

1233     <baselineId>...</baselineId> ?
1234     <snapshotId>...</snapshotId> ?
1235     xs:any
1236   </recordMetadata> ?
1237 </record> *
1238 <instanceId>
1239   <mdrId>xs:anyURI</mdrId>
1240   <localId>xs:anyURI</localId>
1241 </instanceId> +
1242 <additionalRecordType namespace="xs:anyURI"
1243                       localName="xs:NCName" /> *
1244 </item> +
1245 </nodes> *
1246 <edges templateId="xs:ID">
1247   <relationship>
1248     <source>
1249       <mdrId>xs:anyURI</mdrId>
1250       <localId>xs:anyURI</localId>
1251     </source>
1252     <target>
1253       <mdrId>xs:anyURI</mdrId>
1254       <localId>xs:anyURI</localId>
1255     </target>
1256     <record>
1257       xs:any
1258       <recordMetadata>
1259         <recordId>...</recordId> ?
1260         <lastModified>...</lastModified> ?
1261         <baselineId>...</baselineId> ?
1262         <snapshotId>...</snapshotId> ?
1263       </recordMetadata> ?
1264     </record> *
1265     <instanceId>
1266       <mdrId>xs:anyURI</mdrId>
1267       <localId>xs:anyURI</localId>
1268     </instanceId> +
1269     <additionalRecordType namespace="xs:anyURI"
1270                           localName="xs:NCName" /> *
1271   </relationship> +
1272 </edges> *
1273 </queryResult>

```

1274 Each time an item matches an `<itemTemplate>`, an `<item>` element appears inside a `<nodes>`  
1275 element in the `<queryResult>` (unless the `itemTemplate` has the attribute "suppressFromResults" set to  
1276 true). Note that for an item to "match" an `<itemTemplate>` it needs to not just meet the conditions inside  
1277 the `<itemTemplate>` but also any `<relationshipTemplate>` that references the  
1278 `<itemTemplate>` as described in 6.2.2. The `templateId` attribute of the response `<nodes>` element  
1279 containing the item has the same value as the `id` attribute of the corresponding `<itemTemplate>` in the  
1280 original request. If the item matches more than one `<itemTemplate>`, the `<item>` will be contained in  
1281 the `<nodes>` for each `<itemTemplate>` matched by the item that doesn't have the

1282 "suppressFromResults" attribute set to true (each <nodes> element with the appropriate value for its  
1283 templateId attribute).

1284 Similarly, each time a relationship matches a <relationshipTemplate>, a <relationship>  
1285 element appears inside an <edges> element in the <queryResult>. The templateId attribute of this  
1286 element contains the same value as the ID attribute of the <relationshipTemplate> in the original  
1287 request. If the relationship matches more than one <relationshipTemplate>, the <relationship>  
1288 is contained in the <edges> for each <relationshipTemplate> matched by the relationship (each  
1289 one with the appropriate value for its templateId attribute).

1290 If no item is part of the response, there are no <nodes> elements. If no relationship is part of the  
1291 response, there are no <edges> elements.

1292 Items and relationships can contain any number of records. Each is represented by a <record> element.  
1293 Each record element contains one or two child elements. The first child is an element whose QName is a  
1294 recordType supported by the Query Service or a <propertySet> element (see 6.6.1), which would  
1295 contain a subset of the properties of the recordType.. The children of that child are the properties  
1296 associated with the record. The optional second child is a <recordMetadata> element that contains  
1297 information about the record itself.

1298 Items and relationships shall contain at least one <instanceId> element. The instance ID, through a  
1299 combination of two URIs (<mdrId> to represent the MDR that assigned the ID and <localId> to  
1300 uniquely represent the item or relationship inside this MDR), uniquely and globally identifies the item or  
1301 relationship. There can be more than one <instanceId> element, in the case where the item or  
1302 relationship has been reconciled from a more fragmented view.

1303 The <source> child element of a relationship identifies the item that is the source of the relationship. The  
1304 format of this element matches the format of the <instanceId> element on the item.

1305 The <target> child element of a relationship identifies the item that is the target of the relationship. The  
1306 format of this element matches the format of the <instanceId> element on the item.

### 1307 6.6.1 propertySet

1308 A query may use <contentSelector>/<selectedRecordType>/<selectedProperty> or  
1309 <contentSelector>/<xpathSelector> to request a subset of the properties of a record type. If the  
1310 subset omits any mandatory properties, the resulting XML element would not be valid according to its  
1311 schema. In this case, the query processor shall place the requested properties inside a <propertySet>  
1312 element to avoid schema violations.

1313 The pseudo-schema of this element is as follows:

```
1314 <propertySet namespace="xs:anyURI" localName="xs:NCName" >  
1315   xs:any *  
1316 </propertySet>
```

1317 The attributes are:

1318     **@namespace** – The namespace of the QName of the record type.

1319     **@localName** – The localName of the QName of the record type.

1320 The child elements of <propertySet> are each child elements of the record type whose QName is  
1321 constructed from the namespace and localName attributes.

## 1322 6.7 GraphQL Faults

1323 The faults defined in this section are generated if the condition stated in the preamble is met. Faults are  
1324 targeted at a destination endpoint according to the fault-handling rules defined by the Web service  
1325 binding.

1326 The definitions of faults in this section use the following properties:

- 1327 • [Code] The fault code.
- 1328 • [Subcode] The fault subcode.
- 1329 • [Reason] The English language reason element.
- 1330 • [Detail] The detail element. If absent, no detail element is defined for the fault.

### 1331 6.7.1 Unknown Template ID

1332 This fault occurs when a `<relationshipTemplate>` includes an ID that refers to a  
1333 `<sourceTemplate>`, `<targetTemplate>`, or `<intermediateItemTemplate>` that was not included  
1334 in the query.

1335 The properties are as follows:

- 1336 • [Code] Sender
- 1337 • [Subcode] `cmdbf:UnknownTemplateIDFault`
- 1338 • [Reason] The graph template ID was not declared.
- 1339 • [Detail] `<cmdbf:graphId> xs:ID </cmdbf:graphId>`

### 1340 6.7.2 Property Type Mismatch

1341 This fault occurs when the value in a constraint is invalid for the type of the property as defined by the  
1342 schema for the property. For example, this fault occurs when the property is a date and the query  
1343 includes a parameter to compare to the date that is a string that cannot be cast to a date, such as  
1344 "foobar."

1345 The properties are as follows:

- 1346 • [Code] Sender
- 1347 • [Subcode] `cmdbf:InvalidPropertyTypeFault`
- 1348 • [Reason] The property value being compared is not valid.
- 1349 • [Detail] `<cmdbf:propertyName namespace="xs:anyURI" localname="xs:NCName" />`

### 1350 6.7.3 XPath Processing Error

1351 This fault occurs when the XPath expression processing results in an error. See [XPath 2.0](#) for details on  
1352 the `cmdbf:xpathErrorCode`.

1353 The properties are as follows:

- 1354 • [Code] Sender
- 1355 • [Subcode] `cmdbf:XPathErrorFault`
- 1356 • [Reason] The XPath expression was not processed successfully.

- 1357 • [Detail] `<cmdbf:expression> xs:string </cmdbf:expression>`
- 1358 `<cmdbf:xpathErrorCode> [xpath error code] </cmdbf:xpathErrorCode>`

#### 1359 6.7.4 Unsupported Constraint

1360 A constraint element in the template was specified that is not supported by this MDR.

1361 The properties are as follows:

- 1362 • [Code] Receiver
- 1363 • [Subcode] cmdbf:UnsupportedConstraintFault
- 1364 • [Reason] The constraint specified is unsupported.
- 1365 • [Detail] `<cmdbf:constraint namespace="xs:anyURI" localname="xs:NCName" />`

#### 1366 6.7.5 Unsupported Selector

1367 A selector element in the template was specified that is not supported by this MDR.

1368 The properties are as follows:

- 1369 • [Code] Receiver
- 1370 • [Subcode] cmdbf:UnsupportedSelectorFault
- 1371 • [Reason] The selector specified is unsupported.
- 1372 • [Detail] `<cmdbf:selector namespace="xs:anyURI" localname="xs:NCName" />`

#### 1373 6.7.6 Expensive Query Error

1374 The query was valid, but the server determined that the query is too expensive to execute or that it would  
 1375 return a result set that is too large to return. The requestor is invited to retry, using a simpler and/or more  
 1376 constrained query. What constitutes “too expensive” or “too large” is determined by the server.

1377 The properties are as follows:

- 1378 • [Code] Receiver
- 1379 • [Subcode] cmdbf:ExpensiveQueryErrorFault
- 1380 • [Reason] The query in the request is too expensive for the server to process or returns a  
 1381 result set that is too large to return.
- 1382 • [Detail] `xs:any`

#### 1383 6.7.7 Query Error

1384 The query was valid, but there was an error while performing the query. When the query includes an  
 1385 XPath expression, this error may be used to indicate that the specific XPath dialect is not supported.

1386 The properties are as follows:

- 1387 • [Code] Receiver
- 1388 • [Subcode] cmdbf:QueryErrorFault
- 1389 • [Reason] An error occurred while processing the request.
- 1390 • [Detail] `xs:any`



## 1391 7 Registration Service

### 1392 7.1 Overview

1393 The Registration Service is used in push mode federation, as described in 5.3.2.1.

1394 The fundamentals of push mode federation are:

- 1395 • The MDR invokes the Register operation for items or relationships that it wishes to register.  
1396 Each item or relationship shall be associated with at least one record type supported by the  
1397 Registration Service. The MDR may register a subset of the data records it has about any item  
1398 or relationship.
- 1399 • The Registration Service responds with the registration status for each item or relationship  
1400 named in the Register operation. The status is either accepted or declined.
  - 1401 – If the return status is accepted, the Registration Service returns the ID that identifies the  
1402 item or relationship within the Registration Service. For accepted data, the MDR is  
1403 expected to update the Registration Service whenever any of the registered data changes.  
1404 This specification does not stipulate how soon after the data changes the update must  
1405 occur — this would typically be determined by local policy.
  - 1406 – If the return status is declined, the Registration Service presumably does not maintain the  
1407 registration data and no updates to that data are accepted. For previously accepted data, a  
1408 return status of declined indicates that the Registration Service no longer wishes to be  
1409 updated about this item. The client would typically deregister the item's ID or attempt to re-  
1410 register the item, perhaps with new data.
- 1411 • This specification does not stipulate what the Registration Service should or shall do with the  
1412 registered data. The semantics of accepted and declined have meaning only with respect to the  
1413 obligations of the MDR to update the Registration Service when the data changes.
- 1414 • The MDR also uses the Register operation to update registered data. An update may consist of  
1415 any combination of the following actions:
  - 1416 – Changing existing data, such as a property value
  - 1417 – Registering an additional record type for this item or relationship
  - 1418 – Deregistering a previously registered record type for this item or relationship
  - 1419 – The MDR uses the Deregister operation to remove an existing registration for an item or  
1420 relationship. For example, if the item or relationship is deleted, the MDR would typically  
1421 delete its own records and deregister the previous registration. Another example of when  
1422 Deregister would be used is if an administrator decides to stop federating the data about  
1423 this item or relationship, even though the item or relationship still exists and the MDR still  
1424 maintains data about it.
  - 1425 – This specification does not stipulate what the Registration Service should or shall do after a  
1426 Deregister operation.

1427 EXAMPLE:

1428 The following examples show how the Registration Service might handle a deregister operation:

- 1429 – If the Registration Service has the same data from another MDR that this MDR deregisters,  
1430 it might disassociate the data with the deregistering MDR, while maintaining the existing  
1431 data.
- 1432 – If the Registration Service has data from another MDR about the deregistered item or  
1433 relationship, it might delete the deregistered data while maintaining the data from the other  
1434 MDR.

- 1435           – If the Registration Service has the same data from another MDR, but it considers the  
1436           deregistering MDR the authoritative source, it might mark the item or relationship as  
1437           deleted.
- 1438           – If the deregistering MDR is the only source of data about the item or relationship, it might  
1439           delete all knowledge of the item or relationship.

## 1440 7.2 Register

1441 The Register operation is used by an MDR to notify a Registration Service that new items have been  
1442 discovered or updated and data is now available in the MDR.

### 1443 7.2.1 Register Operation

1444 The pseudo-schema for the Register operation is as follows:

```

1445 <registerRequest>
1446   <mdrId>xs:anyURI</mdrId>
1447   <itemList>
1448     <item>
1449       <record>
1450         xs:any
1451         <recordMetadata>...</recordMetadata> ?
1452       </record> *
1453       <instanceId>cmdbf:MdrScopedIdType</instanceId> +
1454       <additionalRecordType namespace="xs:anyURI"
1455         localName="xs:NCName" /> *
1456     </item> +
1457   <itemList> ?
1458   <relationshipList>
1459     <relationship>
1460       <source>cmdbf:MdrScopedIdType</source>
1461       <target>cmdbf:MdrScopedIdType</target>
1462       <record>
1463         xs:any
1464         <recordMetadata>...</recordMetadata> ?
1465       </record> *
1466       <instanceId>cmdbf:MdrScopedIdType</instanceId> +
1467       <additionalRecordType namespace="xs:anyURI"
1468         localName="xs:NCName" /> *
1469     </relationship> +
1470   <relationshipList> ?
1471 </registerRequest>

```

1472 The following subclauses describe additional constraints on the Register operation pseudo-schema.

#### 1473 7.2.1.1 mdrId

1474 The <mdrID> element is the ID of the MDR registering its data. This ID shall be unique among all of the  
1475 MDRs and federating CMDBs that are federated together.

**1476 7.2.1.2 itemList**

1477 The <itemList> element lists the items being registered. The list contains any number of <item>  
1478 elements. However, if the list contains zero <item> elements, including the <itemList> element serves  
1479 no purpose. An <item> should not be repeated in the list.

**1480 7.2.1.3 itemList/item**

1481 The <item> element indicates some or all of the contents of an <item>.

**1482 7.2.1.4 itemList/item/instanceId**

1483 The <instanceId> serves as a unique key for the <item>. There shall be at least one for each  
1484 <item>. The <instanceId> shall contain the values that would select the <item> in a query using an  
1485 <instanceIdConstraint>.

**1486 7.2.1.5 itemList/item/record**

1487 Each <item> contains any number of <record> elements.

1488 The <record> element shall contain exactly one child element of unrestricted type, followed by a  
1489 <recordMetadata> element. The namespace and local name of the first child element together are the  
1490 record type.

1491 The <record> type shall be supported by the Registration Service.

1492 The MDR may support queries for <record> types that it chooses to not federate through the  
1493 Registration Service.

1494 There may be multiple <record> elements. The set of passed elements will be considered a complete  
1495 replacement if the Registration Service already has data from this MDR about this <item>. For example,  
1496 if the MDR had previously registered this <item> with ComputerConfiguration and ComputerAsset  
1497 records, and another registration call is made for the same item with only the ComputerConfiguration  
1498 record, then it will be treated as a deletion of the ComputerAsset record from the federation.

**1499 7.2.1.6 itemList/item/additionalRecordType**

1500 An MDR may support through its query interface record types for an item that are not included in the  
1501 registerRequest message. If so, it may indicate the record types for the item by including one or more  
1502 <additionalRecordType> elements. The <additionalRecordType>/@namespace and  
1503 <additionalRecordType>/@localName attributes together represent the record type. In each  
1504 <item> the same record type should not appear in both an <additionalRecordType> and a  
1505 <record> element.

1506 EXAMPLE: For queries, the MDR may support ComputerIdentification, ComputerConfiguration, and ComputerAsset  
1507 records. If the registerRequest message includes only the ComputerIdentification record contents in the  
1508 <record> element, the MDR may provide in <additionalRecordType> elements the localName  
1509 and namespace URIs for the ComputerConfiguration and ComputerAsset records.

**1510 7.2.1.7 relationshipList**

1511 The <relationshipList> item indicates the list of relationships being registered. The list contains any  
1512 number of <relationship> elements. However, if the list contains zero <relationship> elements,  
1513 including the <relationshipList> element serves no purpose.

1514 **7.2.1.8 relationshipList/relationship**

1515 The <relationship> element includes some or all of the contents of a <relationship>.

1516 **7.2.1.9 relationshipList/relationship/instanceId**

1517 The <instanceId> serves as a unique key for the <relationship>. There shall be at least one  
1518 <instanceId> for each <relationship> element. The <instanceId> shall contain the values that  
1519 would select the <relationship> in a query using an <instanceIdConstraint>.

1520 **7.2.1.10 relationshipList/relationship/source**

1521 The <source> element is the <instanceId> that serves as a unique key for the <item> referenced by  
1522 the source side of a relationship. There shall be exactly one <instanceId> for each  
1523 <relationship>. The <instanceId> shall contain one of the values that would select the source  
1524 <item> in a query using an <instanceIdConstraint>.

1525 **7.2.1.11 relationshipList/relationship/target**

1526 The <target> element is the <instanceId> that serves as a unique key for the <item> referenced by  
1527 the target side of a relationship. There shall be exactly one <instanceId> for each <relationship>.  
1528 The <instanceId> shall contain one of the values that would select the target <item> in a query using  
1529 an <instanceIdConstraint>.

1530 **7.2.1.12 relationshipList/relationship/record**

1531 Each <relationship> contains any number of <record> elements. The <record> type shall be  
1532 supported by the Registration Service.

1533 The MDR may support queries for <record> types that it chooses not to federate through the  
1534 Registration Service.

1535 There may be multiple <record> elements. The set of passed elements will be considered a complete  
1536 replacement if the Registration Service already has data from this MDR about this <relationship>.

1537 EXAMPLE: If the MDR had previously registered this <relationship> with a RunsOn and DependsOn record,  
1538 and another registration call is made for the same item with only the RunsOn record, then it will be  
1539 treated as a deletion of the DependsOn record from the federation.

1540 **7.2.1.13 relationshipList/relationship/additionalRecordType**

1541 An MDR may support through its query interface more record types for a relationship than it federates  
1542 through the Registration Service. If so, it may indicate the record types per relationship instance by  
1543 including one or more <additionalRecordType> elements. The  
1544 <additionalRecordType>/@namespace and <additionalRecordType>/@localName attributes  
1545 together represent the record type. The MDR should not include an <additionalRecordType> if for  
1546 the same record type it includes a <record>.

1547 **7.2.2 Register Response**

1548 The pseudo-schema for the response to a Register operation is as follows:

```
1549 <registerResponse>
1550   <RegisterInstanceResponse>
1551     <instanceId>cmdbf:MdrScopedIdType</instanceId>
1552     <accepted>
1553     <alternateInstanceId>
```

```

1554     cmdbf:MdrScopedIdType
1555     </alternateInstanceId> *
1556   </accepted> ?
1557   <declined>
1558     <reason>xs:string</reason> *
1559   </declined> ?
1560   <RegisterInstanceResponse> *
1561 </registerResponse>

```

1562 The following subclauses describe additional constraints on the Register response pseudo-schema.

### 1563 7.2.2.1 registerInstanceResponse

1564 The <registerInstanceResponse> element indicates the action taken for one item or relationship in  
 1565 the Register request. There can be any number of <registerInstanceResponse> elements. There  
 1566 should be exactly one <registerInstanceResponse> element per item or relationship in the Register  
 1567 request.

### 1568 7.2.2.2 registerInstanceResponse/instanceId

1569 The <instanceId> element is one of the elements from the Register request for an item or relationship.

### 1570 7.2.2.3 registerInstanceResponse/accepted

1571 The <accepted> element indicates that the item or relationship instance was accepted.

1572 Exactly one of either the <accepted> or <declined> elements shall be present.

### 1573 7.2.2.4 registerInstanceResponse/accepted/alternateInstanceId

1574 The <alternateInstanceId> element indicates zero or more elements that contain other IDs by  
 1575 which the item or relationship is known, each one of which is acceptable as a key to select the item or  
 1576 relationship in a query.

### 1577 7.2.2.5 registerInstanceResponse/declined

1578 The <declined> element indicates that the item or relationship instance was declined.

1579 Exactly one of either the <accepted> or <declined> elements shall be present.

### 1580 7.2.2.6 registerInstanceResponse/declined/reason

1581 The <reason> element is zero or more strings that contain the reasons why the registration was  
 1582 declined.

## 1583 7.2.3 Register Operation Faults

1584 The faults defined in this section are generated if the condition stated in the preamble is met. Faults are  
 1585 targeted at a destination endpoint according to the fault-handling rules defined by the Web service  
 1586 binding.

1587 The definitions of faults in this section use the following properties:

- 1588 • [Code] The fault code.
- 1589 • [Subcode] The fault subcode.

- 1590 • [Reason] The English language reason element.
- 1591 • [Detail] The detail element. If absent, no detail element is defined for the fault.

### 1592 7.2.3.1 Invalid Record

1593 The record does not correspond to the schema specifying the data model. This fault occurs when a  
1594 required property does not exist, an extension property is used when the data model does not allow for  
1595 extensions, and so on.

1596 The properties are as follows:

- 1597 • [Code] Sender
- 1598 • [Subcode] cmdbf:InvalidRecordFault
- 1599 • [Reason] The record is invalid.
- 1600 • [Detail] `<cmdbf:recordId> xs:anyURI </cmdbf:recordId>`

### 1601 7.2.3.2 Unsupported Record Type

1602 A record of an unsupported record type was attempted to be registered.

1603 The properties are as follows:

- 1604 • [Code] Sender
- 1605 • [Subcode] cmdbf:UnsupportedRecordTypeFault
- 1606 • [Reason] The record type is not supported.
- 1607 • [Detail] `<cmdbf:recordType namespace="xs:anyURI" localname="xs:NCName" />`

### 1608 7.2.3.3 Invalid MDR ID

1609 The MDR ID specified on an item is not recognized.

1610 The properties are as follows:

- 1611 • [Code] Sender
- 1612 • [Subcode] cmdbf:InvalidMDRFault
- 1613 • [Reason] The MDR is not registered.
- 1614 • [Detail] `<cmdbf:mdrId> xs:anyURI </cmdbf:mdrId>`

### 1615 7.2.3.4 Registration Error

1616 There was a problem with registering the items or relationships.

1617 The properties are as follows:

- 1618 • [Code] Sender
- 1619 • [Subcode] cmdbf:RegistrationErrorFault
- 1620 • [Reason] An error occurred while registering.
- 1621 • [Detail] `<cmdbf:recordId> xs:anyURI </cmdbf:recordId>`

## 1622 7.3 Deregister

1623 The Deregister operation is used by an MDR to notify the Registration Service that the data that an MDR  
 1624 has about an item or relationship will no longer be registered. Each item or relationship needs to be  
 1625 deregistered only once, regardless of the number of <instanceId> elements provided in the register  
 1626 request.

### 1627 7.3.1 Deregister Operation

1628 The pseudo-schema for the Deregister operation is as follows:

```
1629 <deregisterRequest>
1630   <mdrId>xs:anyURI</mdrId>
1631   <itemIdList>
1632     <instanceId>cmdbf:MdrScopedIdType</instanceId> *
1633   <itemIdList> ?
1634   <relationshipIdList>
1635     <instanceId>cmdbf:MdrScopedIdType</instanceId> *
1636   <relationshipIdList> ?
1637 </deregisterRequest>
```

1638 The following subclauses describe additional constraints on the Deregister operation pseudo-schema.

#### 1639 7.3.1.1 mdrId

1640 The <mdrId> is the ID of the MDR deregistering its data. This ID shall be the ID used when the data was  
 1641 registered using the Register request.

#### 1642 7.3.1.2 itemIdList

1643 The <itemIdList> element lists items being deregistered. The list contains any number of  
 1644 <instanceId> elements. However, if the list contains zero <instanceId> elements, including the  
 1645 <itemIdList> element serves no purpose.

#### 1646 7.3.1.3 itemIdList/instanceId

1647 The <instanceId> serves as a key for the <item>. The <instanceId> shall be either the  
 1648 <instanceId> from the Register request or an <alternateInstanceId> from a  
 1649 <registerResponse>. An <instanceId> should not be repeated in the list.

#### 1650 7.3.1.4 relationshipIdList

1651 The <relationshipIdList> element lists the relationships being deregistered. The list contains any  
 1652 number of <instanceId> elements. However, if the list contains zero <instanceId> elements,  
 1653 including the <relationshipIdList> element serves no purpose.

#### 1654 7.3.1.5 relationshipIdList/instanceId

1655 The <instanceId> serves as a key for the <relationship>. The <instanceId> shall be either the  
 1656 <instanceId> from the Register request or an <alternateInstanceId> from a  
 1657 <registerResponse>. An <instanceId> should not be repeated in the list.

## 1658 7.3.2 Deregister Response

1659 The pseudo-schema for the response to a Deregister operation is as follows:

```

1660 <deregisterResponse>
1661   <deregisterInstanceResponse>
1662     <instanceId>cmdbf:MdrScopedIdType</instanceId>
1663     <accepted /> ?
1664     <declined>
1665       <reason>xs:string</reason> *
1666     </declined> ?
1667     <deregisterInstanceResponse> *
1668 </deregisterResponse>

```

1669 The following subclauses describe additional constraints on the Deregister response pseudo-schema.

#### 1670 **7.3.2.1 deregisterInstanceResponse**

1671 The `<deregisterInstanceResponse>` element indicates the action taken for one item or relationship  
 1672 in the Deregister request. There can be any number of `<deregisterInstanceResponse>` elements.  
 1673 There should be exactly one `<deregisterInstanceResponse>` element per item or relationship in the  
 1674 Register request.

#### 1675 **7.3.2.2 deregisterInstanceResponse/instanceId**

1676 The `<instanceId>` element provides the ID from the Deregister request for an item or relationship.

#### 1677 **7.3.2.3 deregisterInstanceResponse/accepted**

1678 The `<accepted>` element indicates that the item or relationship instance was accepted.

1679 Exactly one of either the `<accepted>` or `<declined>` elements shall be present.

#### 1680 **7.3.2.4 deregisterInstanceResponse/declined**

1681 The `<declined>` element indicates that the deregistration of the item or relationship instance was  
 1682 declined. An example of when a Deregister request might be declined is when the Registration Service  
 1683 does not recognize `<instanceId>` in the Deregister request.

1684 Exactly one of either the `<accepted>` or `<declined>` elements shall be present.

#### 1685 **7.3.2.5 deregisterInstanceResponse/declined/reason**

1686 The `<reason>` element includes zero or more strings that contain the reasons that the deregistration was  
 1687 declined.

### 1688 **7.3.3 Deregister Operation Faults**

1689 The faults defined in this section are generated if the condition stated in the preamble is met. Faults are  
 1690 targeted at a destination endpoint according to the fault-handling rules defined by the Web service  
 1691 binding.

1692 The definitions of faults in this section use the following properties:

- 1693 • [Code] The fault code.
- 1694 • [Subcode] The fault subcode.
- 1695 • [Reason] The English language reason element.
- 1696 • [Detail] The detail element. If absent, no detail element is defined for the fault.



1697 **7.3.3.1 Invalid MDR Id**

1698 The MDR ID specified on an item is not recognized.

1699 The properties are as follows:

- 1700 • [Code] Sender
- 1701 • [Subcode] cmdbf:InvalidMDRFault
- 1702 • [Reason] The MDR is not registered.
- 1703 • [Detail] `<cmdbf:mdrId> xs:anyURI </cmdbf:mdrId>`

1704 **7.3.3.2 Deregistration Error**

1705 There was a problem with deregistering the items or relationships.

1706 The properties are as follows:

- 1707 • [Code] Sender
- 1708 • [Subcode] cmdbf:DeregistrationErrorFault
- 1709 • [Reason] An error occurred while deregistering.
- 1710 • [Detail] `<cmdbf:recordId> xs:anyURI </cmdbf:recordId>`

1711 **8 Service Metadata**1712 **8.1 Overview**

1713 The register and query operations defined in this specification have a set of optional features that may be  
 1714 supported by a particular implementation. There are also a number of extensibility points in the  
 1715 specification that allow for the anticipated variability in implementations. One key point of variation is the  
 1716 data model or models supported for record types at a given MDR. Prior to sending register or query  
 1717 messages to an MDR, it may be necessary to inspect the capabilities and data models supported by that  
 1718 particular MDR.

1719 The schema defined in this section includes two elements, `<queryServiceMetadata>` and  
 1720 `<registrationServiceMetadata>`, that can be used to indicate which optional features and data  
 1721 models (or record types) are supported by a particular implementation. It is recommended that each MDR  
 1722 implementation include an instance of the appropriate `<queryServiceMetadata>` and/or  
 1723 `<registrationServiceMetadata>` elements as part of the policies describing the implementation.

1724 An example of how these elements can be incorporated into a WS-Policy `<policy>` element and then  
 1725 associated with the implementation's WSDL binding is provided in ANNEX F.

1726 The subclauses in this section describe the service metadata schema elements  
 1727 `<queryServiceMetadata>` and `<registrationServiceMetadata>` and their contents.

1728 Any MDR supporting the GraphQL operation shall support an `<itemTemplate>` with  
 1729 `<instanceIdConstraint>` query at a minimum. Other query capabilities are optional. The service  
 1730 metadata for the MDR should indicate which optional query capabilities are supported.

## 1731 8.2 Common Service Metadata Elements

1732 Both <queryServiceMetadata> and <registrationServiceMetadata> elements have common  
1733 <serviceDescription> and <recordTypeList> child elements to describe the service and list the  
1734 record types supported by the service. These are described here for later reference.

### 1735 8.2.1 serviceDescription

1736 The required <serviceDescription> element is used to associate the service metadata with the MDR  
1737 that is implementing this service. The <mdrId> is the only required element in the  
1738 <serviceDescription>. The other optional elements in the <serviceDescription>, including an  
1739 extensibility element, allow for further description of the service implementation.

1740 The pseudo-schema of the contents of a <serviceDescription> element is as follows:

```
1741 <serviceDescription>
1742   <mdrId>xs:anyURI</mdrId>
1743   <serviceId>xs:anyURI</serviceId> ?
1744   <description xml:lang="xs:language" xs:string</description> *
1745   xs:any *
1746 </serviceDescription>
```

#### 1747 8.2.1.1 serviceDescription/mdrId

1748 The required <mdrId> is the ID of the MDR that is providing this service.

#### 1749 8.2.1.2 serviceDescription/serviceId

1750 <serviceId> is optional if there is only one instance of this service type (possible service types are  
1751 query or registration) for each MDR ID. If there is more than one instance of a service type for an MDR  
1752 ID, <serviceId> is mandatory so metadata can be correctly associated with the instance.

#### 1753 8.2.1.3 serviceDescription/description

1754 The optional <description> element(s) may be used to describe the service in the languages of choice  
1755 for human consumption. The xml:lang attribute is required. If there are multiple <description>  
1756 elements, it is expected that each will have a different value for xml:lang.

## 1757 8.2.2 recordTypeList

1758 The <recordTypeList> is used to enumerate the elements that are considered valid for use as records  
1759 in the implementation of this service. This list of supported record types may change over time and should  
1760 be kept current by the implementation.

1761 The pseudo-schema of the contents of a <recordTypeList> element is as follows:

```
1762 <recordTypeList>
1763   <recordTypes namespace="xs:anyURI" schemaLocation="xs:anyURI" ? >
1764     <recordType localName="xs:NCName" appliesTo="xs:string">
1765       <superType namespace="..." localName="..." /> *
1766       xs:any *
1767     </recordType> *
1768   </recordTypes> *
1769 </recordTypeList>
```

### 1770 8.2.2.1 recordTypeList/recordTypes

1771 For each different namespace that contains record types supported by the implementation, a  
1772 <recordTypes> element should be included in the metadata that includes the namespace,  
1773 schemaLocation if appropriate, and the list of the element names from that namespace which are  
1774 supported by the implementation as <recordType> elements.

1775       **@namespace** – This mandatory attribute gives the namespace of the data model that includes XML  
1776 elements that correspond to record types supported by the implementation.

1777       **@schemaLocation** – This optional attribute should be included when there is a URI that can be  
1778 resolved to an XML schema representation of the elements belonging to the namespace listed in the  
1779 namespace attribute.

### 1780 8.2.2.2 recordTypeList/recordTypes/recordType

1781 A <recordType> element identifies an element that is supported as a record type in the implementation.  
1782 Each <recordType> element shall be from the namespace identified in the containing <recordTypes>  
1783 element.

1784       **@localName** – The value of this attribute corresponds to the localName of a supported XML  
1785 element that is a valid record type for the implementation.

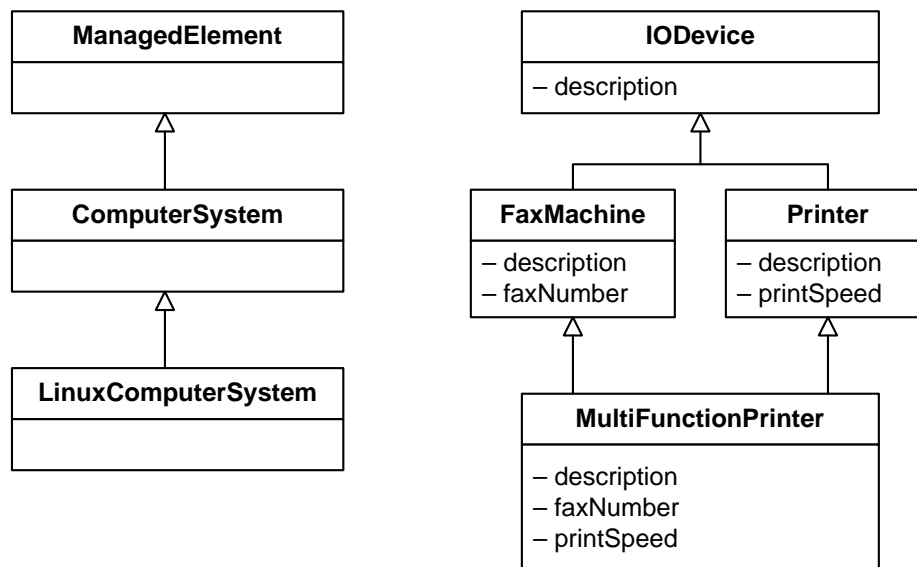
1786       **@appliesTo** – This attribute shall be one of three values indicating whether this element is valid as a  
1787 record in a relationship, item, or both. The values for this attribute are from the enumeration,  
1788 "relationship", "item", or "both".

### 1789 8.2.2.3 recordTypeList/recordTypes/recordType/superType

1790 Record types are often extensions of other record types. A record type is an extension of another record  
1791 type if it has all the properties of the other record type or is the source or target of a relationship that does  
1792 not apply to the other record type. Figure 6 shows two examples of extensions.

1793 In the left example LinuxComputerSystem is an extension of ComputerSystem, which in turn is an  
1794 extension of ManagedElement. LinuxComputerSystem has all the properties of ComputerSystem plus  
1795 adds some other properties specific to Linux. Alternatively or in addition, LinuxComputerSystem could be  
1796 the source or target of a relationship that does not apply to all ComputerSystem instances.

1797 In the right example MultiFunctionPrinter is an extension of both FaxMachine and Printer because it has  
1798 all the properties of FaxMachine and Printer. FaxMachine and Printer are both extensions of IODevice  
1799 because they both have the one property in IODevice.



1800

1801

**Figure 6 – Record Type Extension Examples**

1802 The response to a query for a record type X may contain instances of X or instances of any subtype of X,  
 1803 i.e., any type that declares X to be a super type. A record type is considered a subtype of another record  
 1804 type if all the following are true:

- 1805 • its definition contains all the properties of the super type, and each of these is identically named  
 1806 and typed,
- 1807 • it is valid as the source or target of any relationship that is valid for the super type,
- 1808 • the characterization of the super type applies to the subtype.

1809 A subtype may contain other properties. A record type may have multiple super types.

1810 The `<superType>` element may be used to indicate an extension relationship between record types.

1811 The attributes are:

1812 **@namespace** – The namespace of the QName of the super type.

1813 **@localName** – The localName of the QName of the super type.

### 1814 8.3 queryServiceMetadata

1815 An instance of the `<queryServiceMetadata>` includes the description of the MDR, including the ID of  
 1816 the MDR implementing the Query Service, the supported query capabilities and the supported records, or  
 1817 data model, for the given implementation being modeled.

1818 The pseudo-schema of the contents of a `<queryServiceMetadata>` element is as follows:

```

1819 <queryServiceMetadata>
1820   <serviceDescription> ... </serviceDescription>
1821   <supportedOptionSet>xs:anyURI</supportedOptionSet> *
1822   <queryCapabilities>
1823     <relationshipTemplateSupport depthLimit="xs:boolean"
1824       minimumMaximum="xs:boolean"
  
```

```

1825         xs:anyAttribute /> ?
1826     <contentSelectorSupport recordTypeSelector="xs:boolean"
1827         propertySelector="xs:boolean"
1828         xs:anyAttribute /> ?
1829     <recordConstraintSupport ...> ... </recordConstraintSupport> ?
1830     <xpathSupport>
1831         <dialect>xs:anyURI</dialect> *
1832     </xpathSupport> ?
1833     xs:any *
1834 </queryCapabilities> ?
1835 <recordTypeList> ... </recordTypeList>
1836 xs:any *
1837 </queryServiceMetadata>

```

### 1838 8.3.1 queryServiceMetadata/serviceDescription

1839 The required <serviceDescription> element is used to identify this implementation of the Query  
1840 Service, as previously described.

### 1841 8.3.2 queryServiceMetadata/supportedOptionSet

1842 An option set is a predefined set of query capabilities supported by the service. Each option set is  
1843 identified by a URI. Listing an option set URI in a <supportedOptionSet> element means that the  
1844 service supports all the capabilities that are part of this option set. It doesn't imply that the service does  
1845 not support additional capabilities, just that those that are part of the option set are guaranteed to be  
1846 supported.

1847 If the <queryServiceMetadata> element also contains a <queryCapabilities> section, the  
1848 content of the <queryCapabilities> should list a superset of all the capabilities in all the advertised  
1849 option sets. However, the mere presence of a <supportedOptionSet> element is sufficient to  
1850 advertise the corresponding capabilities, even if a follow-on <queryCapabilities> element fails to list  
1851 them.

1852 In other words, the set of capabilities advertised by the query service is the union of all the capabilities  
1853 that are part of all the listed option sets (using <supportedOptionSet>) and all the capabilities listed in  
1854 the <queryCapabilities> section.

1855 This specification only defines two option sets, described below.

#### 1856 8.3.2.1 Complete Option Set

1857 The URI for this option set is <http://schemas.dmtf.org/cmdmf/1/optionSet/query-complete>.

1858 The complete option set indicate that all query features described in this specification are supported. It is  
1859 equivalent to the following <queryCapabilities> element:

```

1860 <queryCapabilities>
1861     <relationshipTemplateSupport depthLimit="true"
1862         minimumMaximum="true" />
1863     <contentSelectorSupport recordTypeSelector="true"
1864         propertySelector="true" />
1865     <recordConstraintSupport recordTypeConstraint="true"
1866         propertyValueConstraint="true">
1867         <propertyValuesOperators equal="true" less="true"

```

```

1868         lessOrEqual="true" greater="true"
1869         greaterOrEqual="true" contains="true"
1870         like="true" isNull="true" />
1871     </recordConstraintSupport>
1872     <xpathSupport>
1873         <dialect>http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1</dialect>
1874         <dialect>http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath2</dialect>
1875     </xpathSupport>
1876 </queryCapabilities>

```

### 1877 8.3.2.2 Base Option Set

1878 The URI for this option set is <http://schemas.dmtf.org/cmdbf/1/optionSet/query-basic>.

1879 The base option set indicates that all features listed in this specification are supported with the following  
 1880 exceptions:

- 1881 • The @depthLimit attribute is not supported on relationship templates (relationships need to be  
 1882 traversed hop by hop).
- 1883 • The @minimum and @maximum attributes on relationship template are not supported.
- 1884 • Xpath constraints on item templates and relationship templates are not supported.

1885 This option set is equivalent to the following <queryCapabilities> element:

```

1886 <queryCapabilities>
1887     <relationshipTemplateSupport depthLimit="false"
1888         minimumMaximum="false" />
1889     <contentSelectorSupport recordTypeSelector="true"
1890         propertySelector="true" />
1891     <recordConstraintSupport recordTypeConstraint="true"
1892         propertyValueConstraint="true">
1893         <propertyValuesOperators equal="true" less="true"
1894             lessOrEqual="true" greater="true"
1895             greaterOrEqual="true" contains="true"
1896             like="true" isNull="true" />
1897     </recordConstraintSupport>
1898     <xpathSupport/>
1899 </queryCapabilities>

```

### 1900 8.3.3 queryServiceMetadata/queryCapabilities

1901 The <queryCapabilities> element indicates which query techniques described in this specification  
 1902 are supported by this particular implementation of the query operation. The <queryCapabilities>  
 1903 element includes an extensibility element for representing that query extensions beyond the scope of this  
 1904 specification are supported by the implementation.

### 1905 8.3.4 queryServiceMetadata/queryCapabilities/relationshipTemplateSupport

1906 When present, the <relationshipTemplateSupport> element indicates that the query operation of  
1907 the implementation supports queries that include <relationshipTemplate> elements.

1908 **@depthLimit** – The Boolean value of this attribute indicates whether the Query Service  
1909 implementation will process queries with a <depthLimit> element in a  
1910 <relationshipTemplate>.

1911 **@minimumMaximum** – The Boolean value of this attribute indicates whether the Query Service  
1912 implementation will process queries based on the cardinality of relationships as specified by a  
1913 @minimum or @maximum attribute on a <sourceTemplate> or <targetTemplate> element of  
1914 a <relationshipTemplate>.

### 1915 8.3.5 queryServiceMetadata/queryCapabilities/contentSelectorSupport

1916 When present, the <contentSelectorSupport> element indicates that the query operation of the  
1917 implementation supports queries that include <contentSelector> elements.

1918 **@recordTypeSelector** – The Boolean value of this attribute indicates whether the Query Service  
1919 implementation will process queries with <selectedRecordType> specified in the  
1920 <contentSelector> of an <itemTemplate> or <relationshipTemplate>.

1921 **@propertyTypeSelector** – The Boolean value of this attribute indicates whether the Query Service  
1922 implementation will process queries with <selectedProperty> specified in the  
1923 <contentSelector> of an <itemTemplate> or <relationshipTemplate>.

### 1924 8.3.6 queryServiceMetadata/queryCapabilities/recordConstraintSupport

1925 The <recordConstraintSupport> element indicates whether the query implementation will process  
1926 queries that use constraints in the <itemTemplate> or <relationshipTemplate>. The complete  
1927 pseudo-schema of this element is as follows:

```
1928 <recordConstraintSupport recordTypeConstraint="xs:boolean"
1929   propertyValueConstraint="xs:boolean" xs:anyAttribute >
1930   <propertyValueOperators equal="xs:boolean" less="xs:boolean"
1931     lessOrEqual="xs:boolean" greater="xs:boolean"
1932     greaterOrEqual="xs:boolean" contains="xs:boolean"
1933     like="xs:boolean" isNull="xs:boolean" xs:anyAttribute />?
1934 </recordConstraintSupport>
```

1935 **@recordTypeConstraint** – The Boolean value of this attribute indicates whether the Query Service  
1936 implementation will process queries with <recordType> constraints in an <itemTemplate> or  
1937 <relationshipTemplate>.

1938 **@propertyValueConstraint** – The Boolean value of this attribute indicates whether the Query  
1939 Service implementation will process queries with <propertyValue> constraints in an  
1940 <itemTemplate> or <relationshipTemplate>. When <propertyValue> constraints are  
1941 supported the metadata should also indicate which operators are supported by including the  
1942 <propertyValueOperators> element.

### 1943 8.3.7 recordConstraintSupport/propertyValueOperators

1944 The <propertyValueOperators> element is used to indicate which operators are supported by the  
1945 query implementation. There is a mandatory attribute for each operator defined by this specification and  
1946 an extensibility attribute for other operators not defined by this specification.

1947 The Boolean value of each of the following attributes indicates whether the Query Service implementation  
1948 will process queries with a property value operator of the same name as the attribute: @equal, @less,  
1949 @lessOrEqual, @greater, @greaterOrEqual, @contains, @like, and @isNull.

### 1950 **8.3.8 queryServiceMetadata/queryCapabilities/xpathSupport**

1951 The <xpathSupport> element is used to indicate that the query implementation supports the dialects of  
1952 XPath represented by the contained <dialect> elements.

### 1953 **8.3.9 queryServiceMetadata/queryCapabilities/xpathSupport/dialect**

1954 The <dialect> elements indicate which dialects of XPath will be processed by the query  
1955 implementation. The URI used as the value of the dialect should be either of the following:

- 1956 • one of the URIs listed in this specification for XPath dialects
- 1957 • a URI defined by another specification to represent an XPath dialect appropriate for use in the  
1958 query operation defined in this specification

### 1959 **8.3.10 queryServiceMetadata/recordTypeList**

1960 The <recordTypeList> is used to list the record types that can be returned by the Query Service, as  
1961 previously described.

## 1962 **8.4 registrationServiceMetadata**

1963 An instance of the <registrationServiceMetadata> includes the description of the MDR  
1964 implementing the Registration Service, including the ID of the MDR, and the supported records, or data  
1965 model, for the given implementation being modeled.

1966 The pseudo-schema for the contents of a <registrationServiceMetadata> element is as follows:

```
1967 <registrationServiceMetadata>  
1968   <serviceDescription> ... </serviceDescription>  
1969   <recordTypeList> ... </recordTypeList>  
1970   xs:any *  
1971 </registrationServiceMetadata>
```

### 1972 **8.4.1 registrationServiceMetadata/serviceDescription**

1973 The required <serviceDescription> element is used to identify this implementation of the  
1974 Registration Service, as previously described.

### 1975 **8.4.2 registrationServiceMetadata/recordTypeList**

1976 The <recordTypeList> is used to list the record types that can be accepted by the Registration  
1977 Service, as previously described.



## ANNEX A (normative)

### URIs and XML Namespaces

1978  
1979  
1980  
1981

1982 This annex lists the XML namespaces and other URIs defined in this specification.

URI	Description
<a href="http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1">http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath1</a>	Represents an XPath 1 dialect that can be used in queries (see 6.5.1).
<a href="http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath2">http://schemas.dmtf.org/cmdbf/1/dialect/query-xpath2</a>	Represents an XPath 2 dialect that can be used in queries (see 6.5.2).
<a href="http://schemas.dmtf.org/cmdbf/1/optionSet/query-complete">http://schemas.dmtf.org/cmdbf/1/optionSet/query-complete</a>	Represents the set of query service options that contains all possible capabilities (see 8.3.2.1).
<a href="http://schemas.dmtf.org/cmdbf/1/optionSet/query-basic">http://schemas.dmtf.org/cmdbf/1/optionSet/query-basic</a>	Represents a set of query service options that provide basic functionality for a variety of query expressions (see 8.3.2.2).
<a href="http://schemas.dmtf.org/cmdbf/1/action/fault">http://schemas.dmtf.org/cmdbf/1/action/fault</a>	Represents an action in the SOAP binding for faults.
<a href="http://schemas.dmtf.org/cmdbf/1/tns/serviceData">http://schemas.dmtf.org/cmdbf/1/tns/serviceData</a>	Represents the target namespace of the XML schema used by the CMDBf Query and Registration services.
<a href="http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata">http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata</a>	Represents the target namespace of the CMDBf Service Description XML schema.
<a href="http://schemas.dmtf.org/cmdbf/1/tns/query">http://schemas.dmtf.org/cmdbf/1/tns/query</a>	Represents the target namespace in the WSDL for the query service.
<a href="http://schemas.dmtf.org/cmdbf/1/tns/registration">http://schemas.dmtf.org/cmdbf/1/tns/registration</a>	Represents the target namespace in the WSDL for the registration service.

1983

**ANNEX B  
(normative)****CMDB Federation XSD and WSDL**

- 1984
- 1985
- 1986
- 1987
- 1988 Normative copies of the XML schemas for this version of this specification may be retrieved by resolving  
1989 the URLs below.
- 1990 [http://schemas.dmtf.org/cmdbf/1/tns/serviceData/dsp8040\\_1.0.0.xsd](http://schemas.dmtf.org/cmdbf/1/tns/serviceData/dsp8040_1.0.0.xsd)  
1991 [http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata/dsp8041\\_1.0.0.xsd](http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata/dsp8041_1.0.0.xsd)
- 1992 Normative copies of the XML schemas for the current version of this specification (which is this version  
1993 unless it is superseded) may be retrieved by resolving the URLs below.
- 1994 <http://schemas.dmtf.org/cmdbf/1/tns/serviceData/dsp8040.xsd>  
1995 <http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata/dsp8041.xsd>
- 1996 Any `xs:documentation` content in XML schemas for this specification is informative and provided only  
1997 for convenience.
- 1998 Normative copies of the WSDL for the query and registration services described in this version of this  
1999 specification may be retrieved by resolving the URLs below.
- 2000 [http://schemas.dmtf.org/cmdbf/1/tns/query/dsp8043\\_1.0.0.wsdl](http://schemas.dmtf.org/cmdbf/1/tns/query/dsp8043_1.0.0.wsdl)  
2001 [http://schemas.dmtf.org/cmdbf/1/tns/registration/dsp8042\\_1.0.0.wsdl](http://schemas.dmtf.org/cmdbf/1/tns/registration/dsp8042_1.0.0.wsdl)
- 2002 Normative copies of the WSDL for the query and registration services described in the current version of  
2003 this specification (which is this version unless it is superseded) may be retrieved by resolving the URLs  
2004 below.
- 2005 <http://schemas.dmtf.org/cmdbf/1/tns/query/dsp8043.wsdl>  
2006 <http://schemas.dmtf.org/cmdbf/1/tns/registration/dsp8042.wsdl>
- 2007

## ANNEX C (normative)

### Fault Binding to SOAP

2008  
2009  
2010  
2011

2012 Faults may be generated for any CMDBf operation. The bindings of faults for both [SOAP 1.1](#) and  
2013 [SOAP 1.2](#) are described in this annex.

2014 The definitions of faults use the following properties:

- 2015 • [Code] The fault code.
- 2016 • [Subcode] The fault subcode.
- 2017 • [Reason] A language-localized readable description of the error.
- 2018 • [Detail] Optional detail elements. If more than one detail element is defined for a fault,  
2019 implementations shall include the elements in the order that they are specified.

2020 Services that generate CMDBf faults shall set the [Code] property to either "Sender" or "Receiver". These  
2021 properties are serialized into text XML as shown in Table C-1.

2022

**Table C-1 – [Code] Properties**

SOAP Version	Sender	Receiver
SOAP 1.1	S11:Client	S11:Server
SOAP 1.2	S:Sender	S:Receiver

2023 The properties in Table C-1 bind to a [SOAP 1.2](#) fault as follows:

2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044

```

<S:Envelope>
  <S:Header>
    <wsa:Action>
      http://schemas.dmtf.org/cmdbf/1/action/fault
    </wsa:Action>
    <!-- Headers elided for brevity. -->
  </S:Header>
  <S:Body>
    <S:Fault>
      <S:Code>
        <S:Value> [Code] </S:Value>
        <S:Subcode>
          <S:Value> [Subcode] </S:Value>
        </S:Subcode>
      </S:Code>
      <S:Reason>
        <S:Text xml:lang="en"> [Reason] </S:Text>
      </S:Reason>
      <S:Detail>
        [Detail]
        ...
    </S:Fault>
  </S:Body>
</S:Envelope>
```

```
2045     </S:Detail>
2046   </S:Fault>
2047 </S:Body>
2048 </S:Envelope>
```

2049 The properties in Table C-1 bind to a [SOAP 1.1](#) fault as follows when the fault is generated as a result of  
2050 processing a CMDBf request message:

```
2051 <S11:Envelope>
2052   <S11:Header>
2053     <cmdbf:fault>
2054       <cmdbf:faultCode> [Subcode] </cmdbf:faultCode>
2055       <cmdbf:detail> [Detail] </cmdbf:detail>
2056       ...
2057     </cmdbf:fault>
2058     <!-- Headers elided for brevity. -->
2059   </S11:Header>
2060   <S11:Body>
2061     <S11:Fault>
2062       <S11:faultcode> [Code] </S11:faultcode>
2063       <S11:faultstring> [Reason] </S11:faultstring>
2064     </S11:Fault>
2065   </S11:Body>
2066 </S11:Envelope>
```

2067 When binding to a CMDBf operation that supports WS-Addressing, the fault message shall include the  
2068 following action URI as the [action] property:

```
2069     http://schemas.dmtf.org/cmdbf/1/action/fault
```

2070 Fault handling rules for operations using WS-Addressing are defined in section 6 of [WS-Addressing](#)  
2071 [SOAP Binding](#).

2072

## ANNEX D (informative)

### Query Examples

2073  
2074  
2075  
2076

2077 This annex contains two extended GraphQL examples.

#### 2078 D.1 GraphQL Example 1

2079 Let us assume that an MDR contains two types of items (people and computers) and one type of  
2080 relationship (a person "uses" a computer). The following simple query request selects all computers that  
2081 are used by a person located in California:

```
2082 <query>
2083   <itemTemplate id="user">
2084     <recordConstraint>
2085       <recordType namespace="http://example.com/people"
2086         localName="person" />
2087       <propertyValue namespace="http://example.com/people"
2088         localName="state">
2089         <equal>CA</equal>
2090       </propertyValue>
2091     </recordConstraint>
2092   </itemTemplate>
2093
2094   <itemTemplate id="computer">
2095     <recordConstraint>
2096       <recordType namespace="http://example.com/computer"
2097         localName="computer" />
2098     </recordConstraint>
2099   </itemTemplate>
2100
2101   <relationshipTemplate id="usage">
2102     <recordConstraint>
2103       <recordType namespace="http://example.com/computer"
2104         localName="uses" />
2105     </recordConstraint>
2106     <sourceTemplate ref="user" />
2107     <targetTemplate ref="computer" />
2108   </relationshipTemplate>
2109
2110 </query>
```

2111 The detailed syntax and semantics of the XML elements were described in the body of this specification,  
2112 but the following summary describes the items and relationships that are returned by this query:

2113 The `<itemTemplate>` called "user" (line 02) matches all items that:

- 2114 • have a record with a property called "state" (in the `http://example.com/people` namespace) for  
2115 which the value is "CA"
- 2116 • have a record named "person" (defined in the namespace "`http://example.com/people`")

- 2117 • are the source of a relationship that matches the <relationshipTemplate> called "usage"  
2118 (line 11)

2119 The <itemTemplate> called "computer" (line 08) matches all items that:

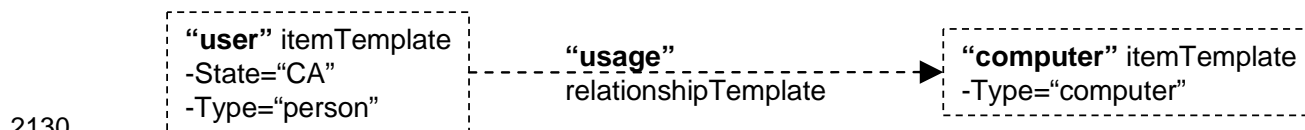
- 2120 • have a record named "computer" (defined in the namespace "http://example.com/computer")
- 2121 • are the target of a relationship that matches the <relationshipTemplate> called "usage"  
2122 (line 11)

2123 The <relationshipTemplate> called "usage" (line 11) matches all relationships that:

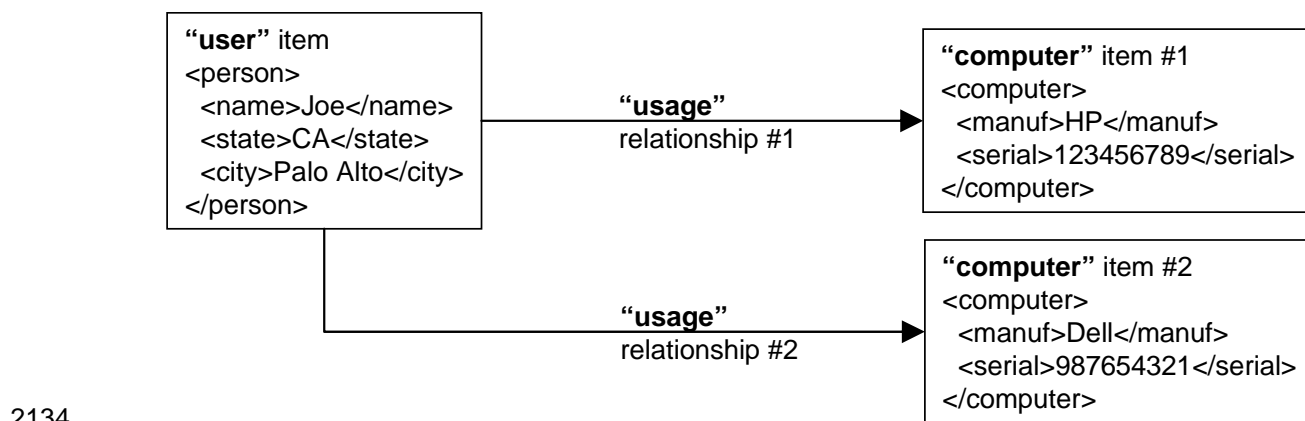
- 2124 • have a record named "uses" (defined in the namespace "http://example.com/computer")
- 2125 • have a source that matches the <itemTemplate> called "user" (line 02)
- 2126 • have a target that matches the <itemTemplate> called "computer" (line 08)

2127 As a result, if a user item does not "use" a computer, it will not be part of the response, whether or not the  
2128 user is located in California.

2129 The following is a graphical representation of the query:



2131 A user in California who happens to "use" two computers is represented in the response by three items  
2132 (one for the user and one for each computer) and two relationships (from the user to each of his or her  
2133 computers). The following is a graphical representation of this response:



2135 In effect, the response contains two graphs (each made of a user, a computer, and the relationship  
2136 between the two) that both meet the constraints of the query graph. In this example, the two graphs in the  
2137 response happen to overlap (they share the same "user"), but in another example they could be disjoint  
2138 (for example, if the second computer were instead "used" by another user also located in California).

2139 If the <relationshipTemplate> element (line 11) were not part of the query, the semantics of the  
2140 query would be very different. The query would return all the items of type "person" that are in California  
2141 and all the items of type "computer". It would not return the relationships between users and computers.  
2142 The existence of these relationships would have no bearing on what items are returned.

2143 The GraphQL operation can also use relationships to qualify instances, even when the result of the  
 2144 query does not include relationships. In the previous example, suppose that we are interested only in the  
 2145 computers used by people in California, not the users themselves. We can add `suppressFromResult=true`  
 2146 to the "user" and "usage" templates in the previous query. The query result is simply the two computers  
 2147 listed above.

```

2148 <query>
2149   <itemTemplate id="user" suppressFromResult="true">
2150     <recordConstraint>
2151       <recordType namespace="http://example.com/people"
2152         localName="person"/>
2153       <propertyValue namespace="http://example.com/people"
2154         localName="state">
2155         <equal>CA</equal>
2156       </propertyValue>
2157     </recordConstraint>
2158   </itemTemplate>
2159   <itemTemplate id="computer">
2160     <recordConstraint>
2161       <recordType namespace="http://example.com/computer"
2162         localName="computer"/>
2163     </recordConstraint>
2164   </itemTemplate>
2165   <relationshipTemplate id="usage" suppressFromResult="true">
2166     <recordConstraint>
2167       <recordType namespace="http://example.com/computer"
2168         localName="uses"/>
2169     </recordConstraint>
2170     <sourceTemplate ref="user"/>
2171     <targetTemplate ref="computer"/>
2172   </relationshipTemplate>
2173 </query>
    
```

2174 **D.2 GraphQL Example 2**

2175 In this example, the data model contains item records of type `ContactInfo` and `ComputerConfig` and  
 2176 relationship records of type "administers". `ComputerConfigs` are related to `ContactInfo` through the  
 2177 "administers" relationship to allow for modeling logic, such as "UserA administers ComputerB."

2178 This example queries the graph of the computers that are administered by "Pete the Lab Tech" and  
 2179 returns all items and relationships involved in this graph. The response shows two computers  
 2180 administrated by one user.

2181 The data the query is executed against are as follows:

2182 **Table D-1 – "User (ContactInfo)" Data**

Name	Phone	employeeNumber
Pete the Lab Tech	111-111-1111	109
Joe the Manager	111-111-4567	12
Frank the CEO	111-111-9999	1

2183

Table D-2 – "Computer (ComputerConfig)" Data

Name	primaryMACAddress	CPUType	assetTag
LabMachineA	00A4B49D2F41	AMD Athlon 64	XYZ9753
LabMachineB	00A4B49D2F42	AMD Athlon 64	XYZ9876
LabMachineC	00A4B49D2H11	Intel Pentium 4	XYZ9900
LabMachineD	00A4B49D2H53	Intel Pentium 4	XYZ9912

2184

Table D-3 – "Administers" Data

"User" Name	"Computer" Name	adminSupportHours
Pete the Lab Tech	LabMachineA	24/7
Pete the Lab Tech	LabMachineB	business hours only
Joe the Manager	LabMachineD	24/7

2185 The following example involves a relationship traversal:

```

2186 <query>
2187   <itemTemplate id="user">
2188     <recordConstraint>
2189       <recordType namespace=http://example.com/people
2190         localName="ContactInfo" />
2191       <propertyValue namespace=http://example.com/people
2192         localName="name">
2193         <equal>Pete the Lab Tech</equal>
2194       </propertyValue>
2195     </recordConstraint>
2196   </itemTemplate>
2197   <itemTemplate id="computer">
2198     <recordConstraint>
2199       <recordType
2200         namespace=http://example.com/computerModel
2201         localName="ComputerConfig" />
2202     </recordConstraint>
2203   </itemTemplate>
2204   <relationshipTemplate id="administers">
2205     <recordConstraint>
2206       <recordType
2207         namespace=http://example.com/computerModel
2208         localName="administers" />
2209     </recordConstraint>
2210     <sourceTemplate ref="user" />
2211     <targetTemplate ref="computer" />
2212   </relationshipTemplate>
2213 </query>

```



2214 The following is a response to the GraphQL query:

```

2215 <queryResult>
2216   <nodes templateId="user">
2217     <item>
2218       <record xmlns:hr="http://example.com/people">
2219         <hr:ContactInfo>
2220           <hr:name>Pete the Lab Tech</hr:name>
2221           <hr:phone>111-111-1111</hr:phone>
2222           <hr:employeeNumber>109</hr:employeeNumber>
2223         </hr:ContactInfo>
2224         <recordMetadata>
2225           <recordId>http://example.com/109/Current</recordId>
2226         </recordMetadata>
2227       </record>
2228       <instanceId>
2229         <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
2230         <localId>http://example.com/PeteTheLabTech</localId>
2231       </instanceId>
2232     </item>
2233   </nodes>
2234   <nodes templateId="computer">
2235     <item>
2236       <record xmlns:comp="http://example.com/computerModel">
2237         <comp:ComputerConfig>
2238           <comp:CPUType>AMD Athlon 64</comp:CPUType>
2239           <comp:assetTag>XYZ9753</comp:assetTag>
2240           <comp:primaryMACAddress>
2241             00A4B49D2F41
2242           </comp:primaryMACAddress>
2243           <comp:name>LabMachineA</comp:name>
2244           ...
2245         </comp:ComputerConfig>
2246         <recordMetadata>
2247           <recordId>
2248             http://example.com/machines/XYZ9753/scanned
2249           </recordId>
2250         </recordMetadata>
2251       </record>
2252       <instanceId>
2253         <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
2254         <localId>http://example.com/machines/XYZ9753</localId>
2255       </instanceId>
2256     </item>
2257     <item>
2258       <record xmlns:comp="http://example.com/computerModel">
2259         <comp:ComputerConfig>
2260           <comp:CPUType>AMD Athlon 64</comp:CPUType>
2261           <comp:assetTag>XYZ9876</comp:assetTag>
2262           <comp:primaryMACAddress>

```

```

2263         00A4B49D2F42
2264     </comp:primaryMACAddress>
2265     <comp:name>LabMachineB</comp:name>
2266     ...
2267 </comp:ComputerConfig>
2268 <recordMetadata>
2269     <recordId>
2270         http://example.com/machines/XYZ9876/scanned
2271     </recordId>
2272 </recordMetadata>
2273 </record>
2274 <instanceId>
2275     <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
2276     <localId>http://example.com/machines/XYZ9876</localId>
2277 </instanceId>
2278 </item>
2279 </nodes>
2280 <edges templateId="administers">
2281     <relationship>
2282         <source>
2283             <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
2284             <localId>http://example.com/PeteTheLabTech</localId>
2285         </source>
2286         <target>
2287             <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
2288             <localId>http://example.com/machines/XYZ9876</localId>
2289         </target>
2290 <record xmlns:foo="http://example.com/computerModel">
2291     <foo:administers>
2292         <foo:adminSupportHours>
2293             business hours only
2294         </foo:adminSupportHours>
2295     </foo:administers>
2296     <recordMetadata>
2297         <recordId>adm10001</recordId>
2298     </recordMetadata>
2299 </record>
2300 <instanceId>
2301     <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
2302     <localId>
2303         http://example.com/administers/PeteTheLabTechToLabMachineB
2304     </localId>
2305 </instanceId>
2306 </relationship>
2307 <relationship>
2308     <source>
2309         <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
2310         <localId>http://example.com/PeteTheLabTech</localId>
2311     </source>

```

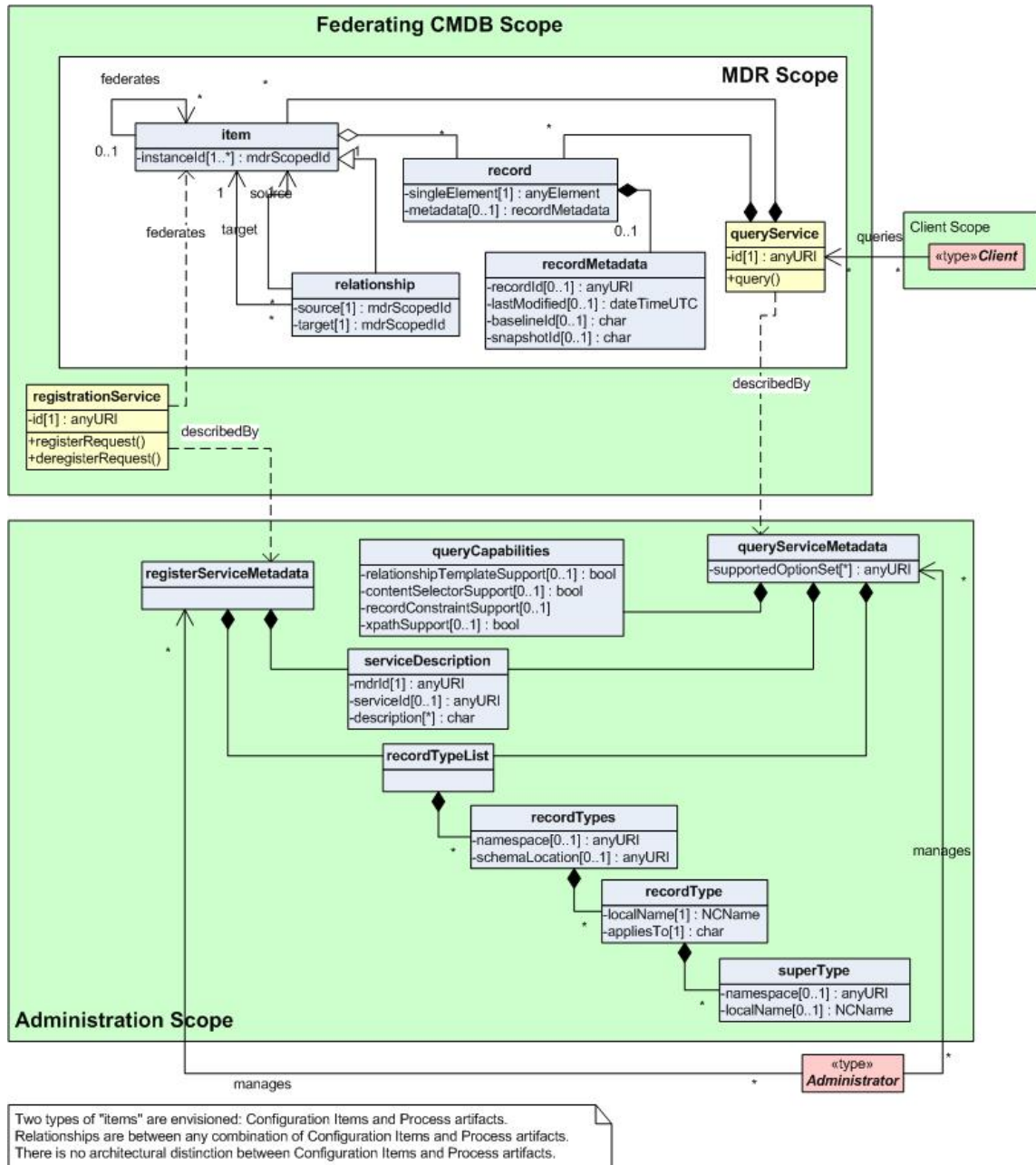
```
2312     <target>
2313         <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
2314         <localId>http://example.com/machines/XYZ9753</localId>
2315     </target>
2316     <record xmlns:foo="http://example.com/computerModel">
2317         <foo:administers>
2318             <foo:adminSupportHours>24/7</foo:adminSupportHours>
2319         </foo:administers>
2320         <recordMetadata>
2321             <recordId>adm10002</recordId>
2322         </recordMetadata>
2323     </record>
2324     <instanceId>
2325         <mdrId>http://testSystem.com/DiscoveryMdr</mdrId>
2326         <localId>
2327             http://example.com/administers/PeteTheLabTechToLabMachineA
2328         </localId>
2329     </instanceId>
2330 </relationship>
2331 </edges>
2332 </queryResult>
```

2333

# ANNEX E (informative)

## Detailed UML Class Diagrams

2334  
2335  
2336  
2337



2338

2339

Figure E-1 – UML Class Diagrams

## ANNEX F (informative)

### Sample WSDL Binding

2340  
2341  
2342  
2343

2344 The following example illustrates how the interfaces defined in this specification should be described in a  
2345 Web service binding that implements the interfaces. This example also illustrates how the CMDBf service  
2346 metadata should be associated with a particular implementation of a CMDBf interface.

2347 As shown below, this query implementation uses [SOAP 1.1](#) over HTTP as the protocol and supports the  
2348 use of WS-Addressing if the message sender uses WS-Addressing for an asynchronous  
2349 request/response. Because this specification does not define specific WS-Addressing actions, the action  
2350 header values for WS-Addressing are determined according to the defaults described in the  
2351 [WS-Addressing 1.0 – WSDL Binding](#) specification.

2352 The queryServiceMetadata element is included in a WS-Policy expression which is included by reference  
2353 in the WSDL binding to the query port type. This particular sample is of a Query Service that supports the  
2354 complete set of record constraint and selector operators defined in the specification. The metadata in the  
2355 sample also shows that XPath1 and XPath 2 are supported by the service.

2356 The metadata for the service also includes the two record types that may be queried at this service, an  
2357 "R\_ComputerSystem" data type, and a "CIM\_CommonDatabase" data type.

2358 The approach to including metadata as a policy in the WSDL is a recommended approach to creating the  
2359 WSDL documentation for the binding implementation as it allows for the file containing the WSDL binding  
2360 to completely describe the interface to the service and the options allowed by this specification.

```
2361 <?xml version='1.0' encoding='UTF-8' ?>
2362 <!--
2363 Copyright © 2009 Distributed Management Task Force, Inc. (DMTF). All rights reserved.
2364 DMTF is a not-for-profit association of industry members dedicated to promoting
2365 enterprise and systems management and interoperability. Members and non-members may
2366 reproduce DMTF specifications and documents provided that correct attribution is
2367 given. As DMTF specifications may be revised from time to time, the particular version
2368 and release date should always be noted. Implementation of certain elements of this
2369 standard or proposed standard may be subject to third party patent rights, including
2370 provisional patent rights (herein "patent rights"). DMTF makes no representations to
2371 users of the standard as to the existence of such rights, and is not responsible to
2372 recognize, disclose, or identify any or all such third party patent right, owners or
2373 claimants, nor for any incomplete or inaccurate identification or disclosure of such
2374 rights, owners or claimants. DMTF shall have no liability to any party, in any manner
2375 or circumstance, under any legal theory whatsoever, for failure to recognize,
2376 disclose, or identify any such third party patent rights, or for such party's
2377 reliance on the standard or incorporation thereof in its product, protocols or testing
2378 procedures. DMTF shall have no liability to any party implementing such standard,
2379 whether such implementation is foreseeable or not, nor to any patent owner or
2380 claimant, and shall have no liability or responsibility for costs or losses incurred
2381 if a standard is withdrawn or modified after publication, and shall be indemnified and
2382 held harmless by any party implementing the standard from any and all claims of
2383 infringement by a patent owner for such implementations. For information about patents
2384 held by third-parties which have notified the DMTF that, in their opinion, such patent
2385 may relate to or impact implementations of DMTF standards, visit
2386 http://www.dmtf.org/about/policies/disclosures.php.
2387 -->
2388
2389 <wsdl:definitions
```

```
2390 targetNamespace="http://schemas.dmtf.org/cmdbf/1/tns/query"
2391 xmlns:cmdbfPort="http://schemas.dmtf.org/cmdbf/1/tns/query"
2392 xmlns:cmdbfMetadata="http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata"
2393 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2394 xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
2395 xmlns:wsp="http://www.w3.org/ns/ws-policy"
2396 xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
2397 xmlns:xs="http://www.w3.org/2001/XMLSchema">
2398
2399 <wSDL:import location="query.wsdl"
2400     namespace="http://schemas.dmtf.org/cmdbf/1/tns/query">
2401 </wSDL:import>
2402
2403 <!-- Subject supports WS-Addressing -->
2404 <wsp:Policy xml:Id="SupportsWSAddressing">
2405     <wsam:Addressing wsp:Optional="true">
2406         <wsp:Policy />
2407     </wsam:Addressing>
2408 </wsp:Policy>
2409
2410
2411 <!-- Subject supports the referenced data model in the operations -->
2412 <wsp:Policy xml:Id="SupportedMetadata">
2413     <queryServiceMetadata
2414         xmlns="http://schemas.dmtf.org/cmdbf/1/tns/serviceMetadata">
2415         <serviceDescription>
2416             <mdrId>CMDBf12345</mdrId>
2417         </serviceDescription>
2418         <queryCapabilities>
2419             <contentSelectorSupport propertySelector="true"
2420                 recordTypeSelector="true" />
2421             <recordConstraintSupport recordTypeConstraint="true"
2422                 propertyValueConstraint="true">
2423                 <propertyValueOperators equal="true" less="true"
2424                     greater="true" lessOrEqual="true"
2425                     greaterOrEqual="true"
2426                     contains="true"
2427                     like="false"
2428                     isNull="false" />
2429             </recordConstraintSupport>
2430             <xpathSupport>
2431                 <dialect>
2432                     http://www.w3.org/TR/1999/REC-xpath-19991116
2433                 </dialect>
2434                 <dialect>
2435                     http://www.w3.org/TR/2007/REC-xpath-20070123
2436                 </dialect>
2437             </xpathSupport>
2438         </queryCapabilities>
```

```

2439
2440     <recordTypeList>
2441         <recordTypes namespace="http://cmdbf.org"
2442             schemaLocation="http://cmdbf.org/common_schemas/R_ComputerSystem.xsd">
2443             <recordType localName="R_ComputerSystem" />
2444         </recordTypes>
2445         <recordTypes
2446             namespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_CommonDatabase"
2447             schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_CommonDatabase.xsd">
2448             <recordType localName="CIM_CommonDatabase" />
2449         </recordTypes>
2450     </recordTypeList>
2451
2452 </queryServiceMetadata>
2453 </wsp:Policy>
2454
2455 <!-- Sample Binding for SOAP 1.1 with WS-Addressing support
2456 -->
2457 <wsdl:binding name="QueryBinding" type="cmdbfPort:QueryPortType">
2458     <soap:binding style="document"
2459         transport="http://schemas.xmlsoap.org/soap/http" />
2460     <wsp:PolicyReference URI="SupportsWSAddressing" />
2461     <wsp:PolicyReference URI="SupportedMetadata" />
2462     <wsdl:operation name="GraphQuery">
2463         <wsdl:input>
2464             <soap:body use="literal" />
2465         </wsdl:input>
2466         <wsdl:output>
2467             <soap:body use="literal" />
2468         </wsdl:output>
2469         <wsdl:fault name="UnknownTemplateID">
2470             <soap:fault name="UnknownTemplateID" use="literal" />
2471         </wsdl:fault>
2472         <wsdl:fault name="InvalidPropertyType">
2473             <soap:fault name="InvalidPropertyType" use="literal" />
2474         </wsdl:fault>
2475         <wsdl:fault name="XPathError">
2476             <soap:fault name="XPathError" use="literal" />
2477         </wsdl:fault>
2478         <wsdl:fault name="UnsupportedConstraint">
2479             <soap:fault name="UnsupportedConstraint" use="literal" />
2480         </wsdl:fault>
2481         <wsdl:fault name="UnsupportedSelector">
2482             <soap:fault name="UnsupportedSelector" use="literal" />
2483         </wsdl:fault>
2484         <wsdl:fault name="QueryError">
2485             <soap:fault name="QueryError" use="literal" />
2486         </wsdl:fault>
2487     </wsdl:operation>

```

```
2488     </wsdl:binding>  
2489  
2490 </wsdl:definitions>
```



**ANNEX G**  
(informative)**Change Log**

Version	Date	Description
1.0.0	2009-06-22	DMTF Standard Release
1.0.1	2010-04-22	DMTF Standard Release – Fixed errors in sections 6.6.1 and 6.7.1

2491  
2492  
2493  
24942495  
2496

2497

## Bibliography

2498 W3C, *Web Services Addressing (WS-Addressing) 1.0: Core*, May 2006,

2499 <http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/>

2500 W3C, *Web Services Addressing 1.0 – SOAP Binding*, May 2006,

2501 <http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509>

2502 W3C, *Web Services Addressing 1.0 – WSDL Binding*, May 2006,

2503 <http://www.w3.org/TR/ws-addr-wsdl/>

2504