



**National Institute of
Standards and Technology**
U.S. Department of Commerce

**NIST Interagency Report 7697
(Second Public Draft)**

1 **Common Platform**
2 **Enumeration: Dictionary**
3 **Specification Version 2.3**
4 **(DRAFT)**

5
6 Paul Cichonski
7 David Waltermire
8 Karen Scarfone

**NIST Interagency Report 7697
(Second Public Draft)**

**Common Platform Enumeration:
Dictionary Specification Version 2.3
(DRAFT)**

Paul Cichonski
David Waltermire
Karen Scarfone

C O M P U T E R S E C U R I T Y

Computer Security Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8930

June 2011



U.S. Department of Commerce

Gary Locke, Secretary

National Institute of Standards and Technology

Dr. Patrick D. Gallagher, Director

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analysis to advance the development and productive use of information technology. ITL's responsibilities include the development of technical, physical, administrative, and management standards and guidelines for the cost-effective security and privacy of sensitive unclassified information in Federal computer systems. This Interagency Report discusses ITL's research, guidance, and outreach efforts in computer security and its collaborative activities with industry, government, and academic organizations.

National Institute of Standards and Technology Interagency Report 7697 (DRAFT)
39 pages (June 2011)

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

45

Acknowledgments

46 The authors, Paul Cichonski and David Waltermire of the National Institute of Standards and Technology
47 (NIST), and Karen Scarfone of Scarfone Cybersecurity wish to thank their colleagues who reviewed
48 drafts of this document and contributed to its technical content. The authors would like to acknowledge
49 Harold Booth of NIST, Christopher McCormick of Booz Allen Hamilton, Seth Hanford of Cisco
50 Systems, Inc., Tim Keanini of nCircle, Kent Landfield of McAfee, Inc., Brant A. Cheikes and Mary
51 Parmelee of the MITRE Corporation, Jim Ronayne of Varen Technologies, Shane Shaffer of G2, Inc.,
52 Joseph L. Wolfkiel of the US Department of Defense, and Adam Halbardier of Booz Allen Hamilton for
53 their insights and support throughout the development of the document.

54

Abstract

55 This report defines the Common Platform Enumeration (CPE) Dictionary version 2.3 specification. The
56 CPE Dictionary Specification is a part of a stack of CPE specifications that support a variety of use cases
57 relating to IT product description and naming. An individual CPE dictionary is a repository of IT product
58 names, with each name in the repository identifying a unique class of IT product in the world. This
59 specification defines the semantics of the CPE Dictionary data model and the rules associated with CPE
60 dictionary creation and management. This report also defines and explains the requirements that IT
61 products and services, including CPE dictionaries, must meet for conformance with the CPE Dictionary
62 version 2.3 specification.

63

Note to Reviewers

64 The authors of this report are particularly interested in reviewers' thoughts and suggestions related to CPE
65 dictionary use cases. The introduction to Section 1 discusses common use cases; however, the list there is
66 not meant to be exclusive, and reviewers are encouraged to submit their ideas for additional ways in
67 which CPE dictionaries could be used.

68

Trademark Information

69 CPE is a trademark of The MITRE Corporation.

70 All other registered trademarks or trademarks belong to their respective organizations.

71

72

Table of Contents

73	1. Introduction	1
74	1.1 Purpose and Scope	2
75	1.2 Audience	2
76	1.3 Document Structure	2
77	1.4 Document Conventions	2
78	2. Definitions and Abbreviations	4
79	2.1 Definitions	4
80	2.2 Abbreviations	5
81	3. Relationship to Existing Specifications and Standards	6
82	3.1 Other CPE Version 2.3 Specifications	6
83	3.2 CPE Version 2.2	6
84	4. Conformance	7
85	4.1 Dictionary Use Conformance	7
86	4.2 Dictionary Creation and Maintenance Conformance	7
87	4.2.1 Official CPE Dictionary	8
88	4.2.2 Extended CPE Dictionaries	8
89	5. CPE Dictionary Data Model	9
90	5.1 The cpe-list Element	9
91	5.2 The generator Element	10
92	5.3 The cpe-item Element	10
93	5.4 The cpe23-item Element	11
94	5.5 The provenance-record Element	12
95	5.6 The deprecation Element	13
96	5.7 Extension Points	14
97	6. Dictionary Creation and Maintenance	15
98	6.1 Acceptance Criteria	15
99	6.1.1 Logical Values and Special Characters	15
100	6.1.2 CPE Name Completeness	16
101	6.1.3 CPE Name Uniqueness	16
102	6.2 CPE Dictionary Deprecation Process	17
103	6.3 CPE Dictionary Provenance Data	19
104	6.4 CPE Dictionary Management Documents	19
105	6.4.1 Dictionary Content Management and Decisions Document	19
106	6.4.2 Dictionary Process Management Document	20
107	7. Dictionary Use	21
108	7.1 Identifier Lookup	21
109	7.2 Dictionary Searching	21
110	7.3 Use of Deprecated Identifier Names	22
111	8. CPE Dictionary Operations and Pseudocode	24
112	8.1 Operations on a CPE Dictionary	24
113	8.1.1 Function get_cpe_items(d)	24

114 8.1.2 Function get_cpe_item_WFN(item)24
 115 8.1.3 Function get(w,a)24
 116 8.1.4 Function is_deprecated(item)24
 117 8.1.5 Function getItem(list, index)24
 118 8.1.6 Function strlen(s)24
 119 8.1.7 Function substr(s,b,e)25
 120 8.2 Acceptance Criteria Pseudocode25
 121 8.3 Dictionary Use Pseudocode27

122 **Appendix A— References32**

123 **Appendix B— Change Log33**

124

125 **List of Figures**

126 Figure 8-1: accept-name function.....26
 127 Figure 8-2: contains-restricted-characters function.....26
 128 Figure 8-3: contains-required-attributes function27
 129 Figure 8-4: matches-more-complete-in-dictionary function.....27
 130 Figure 8-5: dictionary-search function29
 131 Figure 8-6: findSupersetMatches function.....29
 132 Figure 8-7: findSubsetMatches function30
 133 Figure 8-8: findExactMatch function.....31

134

135 **List of Tables**

136 Table 8-1: Description of dictionary-search function.....28
 137 Table 8-2: Description of findSupersetMatches function.....29
 138 Table 8-3: Description of findSubsetMatches function.....30
 139 Table 8-4: Description of findExactMatch function.....30

140

141

142 1. Introduction

143 Common Platform Enumeration (CPE) is a standardized method of describing and identifying classes of
 144 applications, operating systems, and hardware devices present in an enterprise's computing assets. CPE
 145 can be used as a source of information for enforcing and verifying IT management policies relating to
 146 these assets, such as vulnerability, configuration, and remediation policies. IT management tools can
 147 collect information about installed products, identify products using their CPE names, and use this
 148 standardized information to help make fully or partially automated decisions regarding the assets.

149 CPE consists of several modular specifications. Combinations of the specifications work together in
 150 layers to perform various functions. One of these specifications, CPE Dictionary, defines a standardized
 151 method for creating and managing CPE dictionaries. A *dictionary* is a repository of CPE names and
 152 metadata associated with the names. Each CPE name in the dictionary identifies a single class of IT
 153 product in the world. The word "class" here signifies that the object identified is not a physical
 154 instantiation of a product on a system, but rather the abstract model of that product. Although
 155 organizations may use a CPE name to represent either a single product class or a set of multiple product
 156 classes, a CPE dictionary stores only bound forms of well-formed CPE names (WFNs) that identify a
 157 single product class, not a set of product classes. These single product-class WFNs in bound form are
 158 referred to as *identifier names*. An example of a WFN and its bound forms is shown below.

159 WFN:

```
161 wfn: [part="o", vendor="microsoft", product="windows_vista", version="6\.0",
162       update="sp1", edition=NA, language=NA, sw_edition="home_premium",
163       target_sw=NA, target_hw="x64", other=NA]
```

164 WFN bound to a URI:

```
166 cpe:/o:microsoft:windows_vista:6.0:sp1:~::~home_premium::~x64~-
```

167 WFN bound to a formatted string:

```
169 cpe:2.3:o:microsoft:windows_vista:6.0:sp1:-::-:home_premium::-:x64:-
```

170 NIST hosts the Official CPE Dictionary¹, which is the authoritative repository of identifier names. The
 171 goal of the CPE stack of specifications is to provide entities within the IT industry a standardized way to
 172 describe and identify IT products, and the Official CPE Dictionary provides the mechanism to support
 173 this interoperability for product identifiers. The authoritative nature of the Official CPE Dictionary allows
 174 organizations to search for and find identifier names in one centralized place without worrying about
 175 dealing with conflicts between federated dictionaries.

176 Organizations may create their own extended CPE dictionaries, which are used to store identifier names
 177 not present in the Official CPE Dictionary. There are several reasons why extended dictionaries are
 178 needed. For example, an organization may have to create identifier names for proprietary products that
 179 are only useful within that organization, such as internally developed software not found outside the
 180 organization. Another possible reason is that an IT company may want to use identifiers for their
 181 unreleased products that do not yet have official identifier names; once the identifier names have
 182 stabilized, the company would submit them to the Official CPE Dictionary. Finally, an organization may
 183 discover new products in use that do not yet have official identifiers; the organization could create
 184 identifiers, add them to its own extended dictionary, submit them for inclusion in the Official CPE
 185 Dictionary, and use them internally while waiting for their addition to the Official CPE Dictionary.

¹ The Official CPE Dictionary is available at <http://nvd.nist.gov/cpe.cfm>.

186 1.1 Purpose and Scope

187 This document defines the specification for CPE Dictionary version 2.3. This includes formally defining
 188 the concept of a CPE dictionary, the rules and policies relating to dictionary instantiation and
 189 management, and the data model that represents all dictionary concepts and relationships. This document
 190 also establishes the concept of an Official CPE Dictionary, as well as the process for how organizations
 191 can extend the Official CPE Dictionary using extended CPE dictionaries.

192
 193 This report only applies to version 2.3 of CPE Dictionary. All other versions are out of the scope of this
 194 report, as are all CPE specifications other than CPE Dictionary. Operational guidelines, such as suggested
 195 processes for submitting new entries to a dictionary, are also out of the scope of this report.

196 1.2 Audience

197 This report is intended for two main audiences: the creators and maintainers of CPE dictionaries, and IT
 198 management tool developers. Readers of this report should already be familiar with CPE naming and
 199 name matching concepts and conventions, as specified in [CPE23-N] and [CPE23-M].

200 1.3 Document Structure

201 The remainder of this report is organized into the following major sections and appendices:

- 202 • Section 2 defines selected terms and abbreviations used within this specification.
- 203 • Section 3 provides an overview of related specifications or standards.
- 204 • Section 4 defines the conformance rules for this specification.
- 205 • Section 5 defines the CPE dictionary data model.
- 206 • Section 6 provides information and requirements related to CPE dictionary creation and
 207 maintenance.
- 208 • Section 7 defines operations for using a CPE dictionary, such as looking up a particular identifier
 209 name or searching for dictionary entries that belong to a particular set of products.
- 210 • Section 8 provides pseudocode that implements concepts defined in other sections of the
 211 specification.
- 212 • Appendix A lists normative and informative references.
- 213 • Appendix B provides a change log that documents significant changes to major drafts of the
 214 specification.

215 1.4 Document Conventions

216 The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”,
 217 “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be
 218 interpreted as described in Request for Comment (RFC) 2119.²

219 Text intended to represent computing system input, output, or algorithmic processing is presented in
 220 fixed-width Courier font.

221 Normative references are listed in Appendix A. These references use the citation convention of a square
 222 bracket notation containing an abbreviation of the overall reference citation, followed by a colon and

² RFC 2119, “Key words for use in RFCs to Indicate Requirement Levels”, is available at <http://www.ietf.org/rfc/rfc2119.txt>.

223 subsection citation where applicable (e.g., [CPE23-N:7.2] is a citation for the CPE 2.3 Naming
224 specification, Section 7.2).

225 This specification adheres to all rules and conventions defined lower in the CPE stack of specifications.
226 The CPE Naming Specification defines the concept of a Well-Formed CPE Name (WFN) that is a logical
227 representation of a CPE name [CPE23-N:5.1]. Wherever possible, this specification uses WFN
228 representation of CPE names to limit the dependency on any particular CPE name binding. However,
229 readers should keep in mind that CPE dictionaries store bound CPE names, not WFNs.

230 This document uses an abstract pseudocode programming language to specify expected computational
231 behavior. Pseudocode is intended to be straightforwardly readable and translatable into actual
232 programming language statements. Note, however, that pseudocode specifications are not necessarily
233 intended to illustrate efficient or optimized programming code; rather, their purpose is to clearly define
234 the desired behavior, leaving it to implementers to choose the best language-specific design which
235 respects that behavior. In some cases, particularly where standardized implementations exist for a given
236 pseudocode function, we describe the function's behavior in prose.

237 When reading pseudocode the following should be kept in mind:

- 238 • All pseudocode functions are *pass by value*, meaning that any changes applied to the supplied
239 arguments within the scope of the function do not affect the values of the variables in the caller's
240 scope.
- 241 • In a few cases, the pseudocode functions reference (more or less) standard library functions,
242 particularly to support string handling. In most cases semantically equivalent functions can be
243 found in the GNU C library, cf.
244 http://www.gnu.org/software/libc/manual/html_node/index.html#toc_String-and-Array-Utilities.

245 2. Definitions and Abbreviations

246 This section defines selected terms and abbreviations used within the document. This section builds on
 247 the terms defined in the CPE Naming specification [CPE23-N] and the CPE Name Matching specification
 248 [CPE23-M], and does not repeat them here.

249 2.1 Definitions

250 **CPE Dictionary:** A repository of identifier CPE names (WFNs in bound form) and associated metadata.

251 **Deprecated Identifier Name:** An identifier name that is no longer valid because it has either been
 252 replaced by a new identifier name or set of identifier names or was created erroneously.

253 **Dictionary Contributor:** An organization or person that submits new identifier CPE names to a
 254 dictionary for inclusion.

255 **Dictionary Creator:** An organization that instantiates a CPE dictionary that conforms to the guidance
 256 within this specification. A dictionary creator is the organization that is ultimately responsible for the
 257 dictionary.

258 **Dictionary Maintainer:** An organization that manages a CPE dictionary and all processes relating to that
 259 CPE dictionary. In the majority of cases, the organization that serves as the dictionary creator for a
 260 specific CPE dictionary will also serve as the dictionary maintainer. Otherwise, generally the dictionary
 261 maintainer is supporting the dictionary on behalf of the dictionary creator.

262 **Dictionary Management Documents:** A set of documentation that captures the rules and processes
 263 specific to a CPE dictionary.

264 **Dictionary Search:** The process of determining which identifier names within a CPE dictionary are
 265 members of a source name that represents a set of products. The possible outcomes of a dictionary search
 266 are Superset Match, Subset Match, and No Match.

267 **Dictionary User:** An organization, individual, product, or service that consumes a CPE dictionary for any
 268 purpose.

269 **Extended CPE Dictionary:** A dictionary that an organization may create to house identifier names not
 270 found in the Official CPE Dictionary.

271 **Identifier Lookup:** The process of determining if a single identifier name exists in a CPE dictionary. The
 272 possible outcomes of an identifier lookup are Match and No Match.
 273

274 **Identifier CPE Name:** A bound representation of a CPE WFN that uniquely identifies a single product
 275 class. Also referred to as an “identifier name”.

276 **Known Data:** A category of information that may be present within an attribute of a CPE name. Known
 277 data represents any meaningful value about a product (e.g., “sp1”, “2.3.4”, “pro”, NA), but does not
 278 include the logical value ANY.
 279

280 **Official CPE Dictionary:** The authoritative repository of identifier names, which is hosted by NIST.

281 **Official Identifier CPE Name:** Any bound representation of a CPE WFN that uniquely identifies a
282 single product class and is contained within the Official CPE Dictionary.

283 **2.2 Abbreviations**

284	CPE	Common Platform Enumeration
285	IR	Interagency Report
286	IT	Information Technology
287	ITL	Information Technology Laboratory
288	NIST	National Institute of Standards and Technology
289	RFC	Request for Comment
290	URI	Uniform Resource Identifier
291	WFN	Well-Formed CPE Name
292	XML	eXtensible Markup Language
293		

DRAFT

294 **3. Relationship to Existing Specifications and Standards**

295 This section explains the relationships between this specification and related specifications or standards.

296 **3.1 Other CPE Version 2.3 Specifications**

297 CPE version 2.3 was constructed using a modular, stack-based approach, with each major component
298 defined in a separate specification. Functional capabilities are built by layering these modular
299 specifications. This architecture opens opportunities for innovation, as novel capabilities can be defined
300 by combining only the needed specifications, and the impacts of change can be better compartmentalized
301 and managed.

302
303 The CPE Dictionary version 2.3 specification builds upon the CPE Naming version 2.3 [CPE23-N] and
304 CPE Name Matching version 2.3 [CPE23-M] specifications to define the concept of a CPE dictionary.

305 **3.2 CPE Version 2.2**

306 The CPE version 2.3 specifications, including this specification, collectively replace [CPE22]. CPE
307 version 2.3 is intended to provide all the capabilities made available by [CPE22] while adding new
308 features suggested by the CPE user community.

309
310 The primary differences between CPE Dictionary versions 2.2 and 2.3 include:

- 311 • Updated deprecation logic that includes one-to-many CPE deprecation
 - 312 • Updated change history and provenance data requirements
 - 313 • Built-in one-to-one mapping between CPE version 2.2 and version 2.3 names; the version 2.3
314 name is embedded in the version 2.2 name so that the instance document will validate against the
315 version 2.2 schema.
- 316

317 **4. Conformance**

318 Products and organizations may want to claim conformance with this specification for a variety of
 319 reasons. For example, a product may want to assert that it uses official identifier names properly.
 320 Organizations may want to claim conformance if they are creating and/or maintaining CPE dictionaries.

321 This section provides the high-level requirements that a product or organization must meet if they are
 322 seeking conformance with this specification. The majority of the requirements listed in this section
 323 reference other sections in this document that fully define the high-level requirement.

324 **4.1 Dictionary Use Conformance**

325 This section contains the set of requirements for IT products asserting conformance with this specification
 326 for dictionary use. Products may claim conformance with this specification to show that the product uses
 327 identifier names properly. All products claiming conformance with this specification **MUST** adhere to the
 328 following requirements:

- 329 1. Products using CPE names as identifiers **MUST** only use identifier names that meet the criteria
 330 defined in Section 6.1. Products using identifier names **SHOULD** only use official identifier
 331 names, which are located in the Official CPE Dictionary. If an official identifier name is not
 332 available, the product **MUST** use an identifier name contained within an extended CPE dictionary
 333 to which it has access.
- 334 2. Products using a CPE dictionary **MUST** adhere to the requirements in Section 7.1 and 7.2 for
 335 identifier lookups and dictionary searches.
- 336 3. When a product consumes or outputs an identifier name, that product **MUST** first determine if the
 337 identifier name is deprecated and act appropriately if it is, in compliance with the process defined
 338 in Section 7.3. This requirement **MAY** be ignored if the product is intending to communicate
 339 information about deprecated identifier names.

340 **4.2 Dictionary Creation and Maintenance Conformance**

341 Organizations creating or maintaining CPE dictionaries may claim conformance with this specification if
 342 the dictionary meets the requirements in this section. This section first provides the generic requirements
 343 for all dictionaries, then provides requirements specific to the dictionary type (i.e. Official CPE
 344 Dictionary or extended CPE dictionary).

- 345 1. The data within a CPE dictionary **MUST** adhere to the data model defined in Section 5; this
 346 includes all entries in the dictionary containing the mandatory elements defined in Section 5.
- 347 2. Organizations creating/maintaining a CPE dictionary **MAY** extend the CPE dictionary data model
 348 defined in Section 5. These organizations **MUST** use only the extension points defined in Section
 349 5.7 when extending the CPE dictionary data model.
- 350 3. All entries in a CPE dictionary **MUST** adhere to the minimum set of identifier name acceptance
 351 criteria defined in Section 6.1.
- 352 4. A CPE dictionary **MUST** adhere to the identifier name deprecation process defined in Section
 353 6.2.

354 5. A CPE dictionary **MUST** capture the identifier name provenance data defined in Section 6.3.

355 The following subsections provide additional requirements specific to each dictionary type.

356 4.2.1 **Official CPE Dictionary**

357 Additional requirements for the Official CPE Dictionary are:

- 358 1. The Official CPE Dictionary **SHOULD NOT** further restrict the acceptance criteria defined in
359 Section 6.1 because doing so would require all extended CPE dictionaries to implement the same
360 restrictions.
- 361 2. The Official CPE Dictionary **MUST** house its own set of management documents as specified in
362 Section 6.4.

363 4.2.2 **Extended CPE Dictionaries**

364 Additional requirements for extended CPE dictionaries are:

- 365 1. Organizations **MAY** use extended CPE dictionaries to store identifier names for proprietary
366 products.
- 367 2. Organizations **MAY** use extended CPE dictionaries to store identifier names not yet found in the
368 Official CPE Dictionary, but these organizations **SHOULD** submit these new identifier names to
369 the Official CPE Dictionary as official identifier names.
- 370 3. An extended CPE dictionary **SHOULD** house its own set of management documents as specified
371 in Section 6.4. If a dictionary does not have its own documents, the dictionary creator or
372 maintainer **SHOULD** reference the Official CPE Dictionary Management Documents as
373 applicable to the extended dictionary.
- 374 4. An extended CPE dictionary **MUST** adhere to all acceptance criteria restrictions implemented in
375 the Official CPE Dictionary. An extended CPE dictionary **MAY** further restrict the Official CPE
376 Dictionary acceptance criteria, but **MUST** not conflict with them.
- 377 5. An extended CPE dictionary **SHOULD NOT** contain identifier names that conflict with the
378 Official CPE Dictionary. If an extended dictionary does contain conflicting names, the extended
379 dictionary's maintainers **SHOULD** try to resolve the conflict with the Official Dictionary.
- 380 6. An extended CPE dictionary **MUST NOT** include any names that are not unique with respect to
381 itself or to the Official CPE Dictionary. Section 6.1.3 defines CPE name uniqueness.

382

383 5. CPE Dictionary Data Model

384 This section defines the data model that all CPE dictionaries MUST implement. The data model does not
 385 prescribe a specific binding or implementation; it merely describes the data that is required to support the
 386 technical use cases. Any CPE dictionary binding MUST implement all data model elements and
 387 properties defined in this section. Additionally, any CPE dictionary binding MAY extend this data model,
 388 but extensions MUST only occur at the valid extension points defined in Section 5.7.

389
 390 A CPE dictionary is a collection of identifier names and metadata associated with these identifiers. To
 391 support this, the CPE Dictionary data model revolves around one core element called *cpe-item* that holds
 392 all the information relating to a single identifier name. The *cpe-item* element has evolved out of the CPE
 393 Dictionary 2.2 XML Schema to support backwards compatibility. The *cpe-item* element contains a *cpe23-*
 394 *item* element that captures all CPE 2.3 specific data, including provenance data and an upgraded
 395 deprecation system.

396 This data model makes special accommodations to ensure that existing version 2.2 CPE dictionaries are
 397 forward compatible with the CPE Dictionary 2.3 XML schemas. Also, any XML schema based bindings
 398 generated from this data model can produce instance data that validates against the CPE Dictionary 2.2
 399 XML schema, if the *cpe23-item* element is used as the “any” element in the CPE Dictionary 2.2 XML
 400 schema and the *title* element is always specified for each *cpe-item* element. To support this, the CPE
 401 dictionary’s XML specification is split into two schemas: the main schema corresponds to the CPE
 402 Dictionary 2.2 XML schema (with the exception of making the *title* element optional instead of
 403 mandatory), and the schema extension contains the CPE Dictionary 2.3 specific elements and attributes.
 404

405 The tables below define the elements of the data model and their properties. Several of the elements allow
 406 organizations to supply additional information that is not covered by these properties. However, such
 407 information is not part of the official CPE Dictionary data model, and there are no requirements for
 408 dictionary users to access or process this information.

409 In the tables below, types prefixed with “cpe_dict:” are from the CPE Dictionary 2.3 XML schema,
 410 “cpe_dict_ext:” prefixes indicate types from the CPE Dictionary 2.3 XML schema extension, and
 411 “cpe_name” prefixes indicate types from the CPE Naming 2.3 XML schema. These schemas, which are
 412 the authoritative XML binding definitions, can be found at [http://scap.nist.gov/schema/cpe/2.3/cpe-](http://scap.nist.gov/schema/cpe/2.3/cpe-dictionary_2.3.xsd)
 413 [dictionary_2.3.xsd](http://scap.nist.gov/schema/cpe/2.3/cpe-dictionary-extension_2.3.xsd), http://scap.nist.gov/schema/cpe/2.3/cpe-dictionary-extension_2.3.xsd, and
 414 http://scap.nist.gov/schema/cpe/2.3/cpe-naming_2.3.xsd, respectively.

415 5.1 The cpe-list Element

416 The cpe-list element contains all of the dictionary entries and dictionary metadata. The table below
 417 describes the cpe-list element’s properties.
 418

Property	Type	Count	Description
generator (element)	cpe_dict:GeneratorType	0-1	Information related to the generation of the document, including the dictionary schema version, the time of the dictionary’s generation, and the name and version of the application used to generate it.
cpe-item (element)	cpe_dict:ItemType	1-n	A container for a single dictionary entry (identifier name) and its metadata.

Property	Type	Count	Description
	xsd:any	0-n	Additional elements that are not part of the CPE namespace. These contain additional information that a user can choose to use or not, but this information is not required to be used or understood.

419
420
421

The rest of Section 5 provides additional information on the generator and cpe-item elements.

422 5.2 The generator Element

423 The generator element contains information about the generation of the dictionary file. The properties of
424 the generator element are listed below.
425

Property	Type	Count	Description
product_name (element)	xsd:string	0-1	The name of the application used to generate the file.
product_version (element)	xsd:string	0-1	The version of the application used to generate the file.
schema_version (element)	xsd:decimal	1	The version of the schema that the document was written in and that should be used for validation.
timestamp (element)	xsd:dateTime	1	When the file was generated.
	xsd:any	0-n	Additional elements that are not part of the CPE namespace. These contain additional information that a user can choose to use or not, but this information is not required to be used or understood.

426

427 5.3 The cpe-item Element

428 The cpe-item element contains all of the information for a single dictionary entry (identifier name),
429 including metadata. The properties of the cpe-item element are defined below. Organizations can supply
430 additional information not covered by these properties, but such information is not part of the official
431 CPE dictionary data model.
432

Property	Type	Count	Description
name (attribute)	cpe-name:cpe22Type	1	The identifier name bound in CPE 2.2 format.
deprecated (attribute)	xsd:boolean	0-1	Whether or not the name has been deprecated. Default value is false.
deprecated_by (attribute)	cpe-name:cpe22Type	0-1	The name that deprecated this name, bound in CPE 2.2 format.
deprecation_date (attribute)	xsd:dateTime	0-1	When the name was deprecated.
title (element)	cpe_dict:TextType	0-n	Human readable title of the name. To support uses intended for multiple languages, the title element supports the xml:lang attribute. At most one title element can appear for each language.

Property	Type	Count	Description
notes (element)	cpe_dict:NotesType	0-n	Optional descriptive material. Only one notes element should be used per language.
references (element)	cpe_dict:ReferencesType	0-1	External references to additional descriptive material. Each reference consists of a piece of text (intended to be human-readable) and a URI (intended to be a URL pointing to a real resource).
check (element)	cpe_dict:CheckType	0-n	Calls out a check, such as an OVAL definition, that can confirm or reject an IT system as an instance of the named platform. Includes a check system specification URI (e.g., the URI for a particular version of OVAL), a check identifier, and an optional external file reference (for example, a pointer to the file where the check is defined).
cpe23-item (element)	cpe_dict_ext:itemType	1	Element that captures all CPE 2.3 specific data including the CPE 2.3 formatted string binding of the name, provenance data, and deprecation data.
	xsd:any	0-n	Additional elements that are not part of the CPE namespace. These contain additional information that a user can choose to use or not, but this information is not required to be used or understood.

433

434 **5.4 The cpe23-item Element**

435 The cpe23-item element contains all CPE 2.3 specific data related to a given identifier name.

436

Property	Type	Count	Description
name (attribute)	cpe_dict_ext:namePattern	1	The identifier name bound to a formatted string [CPE23-N:6.2].
provenance-record (element)	cpe_dict_ext:provenanceRecordType	0-1	Container for all provenance information for the given identifier name.
deprecation (element)	cpe_dict_ext:deprecationType	0-n	Element holding one or more deprecation entries for the given identifier name. It is possible for a single identifier name to have multiple depreciations that occur at different time periods.

Property	Type	Count	Description
	xsd:any	0-n	Additional elements that are not part of the CPE namespace. These contain additional information that a user can choose to use or not, but this information is not required to be used or understood.

437

438 5.5 The provenance-record Element

439 The provenance-record element contains the provenance information for the given identifier name.

440

Property	Type	Count	Description
submitter (element)	cpe_dict_ext:organizationType	1	The organization responsible for submitting the identifier name.
authority (element)	cpe_dict_ext:organizationType	1-n	The authority responsible for endorsing the identifier name. Multiple authorities may endorse the same identifier name.
change-description (element)	cpe_dict_ext:changeDescriptionType	1-n	A description of any changes made to the identifier name or associated metadata.
	xsd:any	0-n	Additional elements that are not part of the CPE namespace. These contain additional information that a user can choose to use or not, but this information is not required to be used or understood.

441

442 The submitter and authority elements have the same structure:

443

Property	Type	Count	Description
system-id (attribute)	xsd:anyURI	1	Unique URI representing the organization.
name (attribute)	xsd:token	1	Human readable name of the organization.
date (attribute)	xsd:dateTime	1	The date the organization performed an action relative to an identifier name. For example, the date the organization submitted or endorsed a particular identifier name.
description (element)	xsd:token	0-1	A high-level description of the organization acting as the submitter or authority.

444

445 The change-description element has the following structure:
446

Property	Type	Count	Description
change-type (attribute)	cpe_dict_ext:changeTypeType	1	The type of change that occurred. The possible values are listed below the table.
date (attribute)	xsd:dateTime	0-1	When the change occurred.
evidence-reference (element)	cpe_dict_ext:evidenceReferenceType	0-1	Supporting evidence for any change to a name or associated metadata, including a link to external information relating to the change.
comments (element)	xsd:token	0-1	Comments explaining the rationale for the change.

447
448 The possible values for the change-type attribute are:

- 449 • ORIGINAL_RECORD: Used when the name is first added to the dictionary.
- 450 • AUTHORITY_CHANGE: Used when the authority behind the name is modified.
- 451 • DEPRECATION: Used when the name is first deprecated.
- 452 • DEPRECATION_MODIFICATION: Used when additional deprecation entries are recorded for a
453 deprecated name.

454
455 The evidence-reference element contains a URI, which references a specific piece of evidence, and an
456 evidence attribute, which has one of the following possible values:

- 457 • CURATOR_UPDATE: The curator of the dictionary discovered information that led to a change.
- 458 • VENDOR_FIX: The vendor of the product identified in the name released or submitted
459 information that led to a change.
- 460 • THIRD_PARTY_FIX: A third party released or submitted information that led to a change.

461 5.6 The deprecation Element

462 A deprecation element (from the cpe23-item element) contains the deprecation information for a specific
463 deprecation of a given identifier name. If a name is deprecated multiple times, there would be a separate
464 instance of the deprecation element for each deprecation.

465

Property	Type	Count	Description
date (attribute)	xsd:dateTime	0-1	When the deprecation occurred.
deprecated-by (element)	cpe_dict_ext:deprecatedInfoType	1-n	The list of names that deprecated the identifier name. The names in the list do not have to be identifier names; they may contain wildcards and represent sets of products.

466
467 The deprecated-by element has the following structure:
468

Property	Type	Count	Description
name (attribute)	cpe_name:cpe23Type	0-1	The name that is deprecating the identifier name.
type (attribute)	cpe_dict_ext:deprecationTypeType	1	The type of deprecation.

469

470 The possible values for the type attribute are:

- 471 • NAME_CORRECTION: Deprecation is of type Identifier Name Correction
- 472 • NAME_REMOVAL: Deprecation is of type Identifier Name Removal
- 473 • ADDITIONAL_INFORMATION: Deprecation is of type Additional Information Discovery

474 **5.7 Extension Points**

475 Organizations may need to capture data not defined in the CPE Dictionary data model. Any organization
476 serving as a CPE dictionary creator or maintainer MAY extend the cpe23-item, cpe-list, and provenance-
477 record elements to capture additional, organization specific data. If organizations extend these elements,
478 this extension MUST only occur by adding additional properties to these elements to capture different
479 types of data, or by restricting the values for specific properties. Organizations MUST NOT define rules
480 conflicting with the properties already defined for these elements.

481

DRAFT

482 **6. Dictionary Creation and Maintenance**

483 This section provides information and requirements related to creating and maintaining CPE dictionaries.
 484 It defines requirements for valid identifier names within dictionaries and explains the process to be
 485 followed when an identifier name needs to be replaced or removed from a dictionary. This section also
 486 briefly discusses requirements for capturing identifier name provenance data. Finally, this section
 487 explains CPE dictionary management documents and provides related requirements.

488 **6.1 Acceptance Criteria**

489 A CPE dictionary holds a collection of identifier names that serve to identify classes of products. A
 490 dictionary maintainer **MUST** only permit valid identifier names in the dictionary.³ An identifier name is
 491 valid if it meets the acceptance criteria defined in the following sub-sections, which ensure a certain
 492 degree of interoperability across all CPE dictionaries. The criteria focus on ensuring that each identifier
 493 name within a CPE dictionary identifies a single product class. Dictionary maintainers **MAY** further
 494 restrict the acceptance criteria defined in this section, but **MUST NOT** relax the criteria. The maintainers
 495 of the Official CPE Dictionary **SHOULD NOT** further restrict the criteria since that would require all
 496 extended dictionaries to implement the same restrictions.

497 **6.1.1 Logical Values and Special Characters**

498 The CPE Naming specification reserves a set of logical values and special characters for use within a CPE
 499 attribute value [CPE23-N:5.3]. The CPE Dictionary specification permits a subset of these to be contained
 500 within identifier names.

501 **6.1.1.1 NA Logical Value**

502 The NA value within a CPE attribute signifies that the attribute is Not Applicable. For example, a vendor
 503 may distribute the first release of a product that contains no update, but later releases an update to this
 504 first release in the form of a service pack. This example contains two distinct products, the first product
 505 that contains no update, and the second product that does contain an update. The identifier name created
 506 to represent the first release of the product should contain an update attribute with a value of NA (i.e.,
 507 update=NA) since the update attribute is known to be not applicable to the product. The identifier name
 508 created to represent the second release of the product should contain an update attribute with a value of
 509 sp1 (i.e., update="sp1").

510 Identifier names **MUST NOT** use the NA logical value as the part, vendor, or product attribute value, but
 511 **MAY** use the NA logical value for any other CPE attributes' values to indicate that the attributes are not
 512 applicable.

513 **6.1.1.2 ANY Logical Value**

514 The use of the ANY logical value for a CPE attribute value signifies that there was not enough
 515 information to populate the specific CPE attribute at the time of name creation. Identifier names **MUST**
 516 **NOT** use the ANY logical value as the part, vendor, product, or version attribute value, but **MAY** use the
 517 ANY logical value for any other CPE attributes' values to indicate a lack of information.

³ Pseudocode that implements this rule is found in Section 8.2, in the function `accept-name`.

518 It is common for the amount of information known about a product to grow over time. To support this, it
 519 is necessary to evolve the identifier name itself by creating a new, more specific identifier name and
 520 deprecating the old identifier name. Section 6.2 defines the formal deprecation process for tracking
 521 changes to identifier names within a dictionary.

522 **6.1.1.3 Special Characters**

523 The CPE Naming specification reserves a set of special characters for use within a CPE attribute value
 524 [CPE23-N:5.3.2]. The CPE Name Matching specification assigns specific meaning to a subset of these
 525 special characters: the asterisk, which is a multi-character wildcard, and the question mark, which is a
 526 single-character wildcard [CPE23-M:5]. This wildcard concept works well when creating expressions to
 527 represent sets of product classes, but not for attempting to identify a single product class. Therefore,
 528 identifier names **MUST NOT** contain any asterisk or question mark characters within attribute values
 529 unless those characters are quoted. However, CPE dictionaries **MAY** contain these values within names
 530 not serving as identifiers (e.g., within deprecation logic). CPE names containing these values are not valid
 531 identifier names.⁴

532 **6.1.2 CPE Name Completeness**

533 Each identifier name **MUST** contain known data (e.g., not the ANY logical value) for the part, vendor,
 534 product, and version attributes' values. This ensures that each identifier name contains the minimum
 535 amount of data required to identify a unique product class. The ANY logical value does not increase the
 536 completeness of an identifier name; only known data can increase the identifier name completeness.
 537 Known data refers to any value that represents a distinct aspect of a product (e.g. "sp1", "2.0",
 538 "Microsoft", "ios"). For attributes other than the part, vendor, and product attributes, known data includes
 539 the NA logical value since it represents the known fact that absolutely no data exists for a specific CPE
 540 attribute.

541 **6.1.3 CPE Name Uniqueness**

542 Every identifier name **MUST** be unique in the sense that it **MUST NOT** encompass a more complete
 543 identifier name within the dictionary. In terms of the set relations and algorithms in the CPE Name
 544 Matching specification [CPE23-M:6], this means that the identifier name **SHALL NOT** be a **SUPERSET**
 545 of any identifier name within the dictionary unless the two names are the same (**EQUAL**).⁵ Before adding
 546 a new identifier name to a dictionary, the dictionary maintainer **MUST** use CPE name matching to
 547 determine CPE uniqueness. If the dictionary is an extended dictionary, both the extended dictionary and
 548 the Official CPE Dictionary **MUST** be checked.

549 The two scenarios that **MUST** be checked for are:

- 550 1. New name too general (the new name is a **SUPERSET** of the old name, but the new name is not
 551 **EQUAL** to the old name). The creator of a new name may not have had the full set of
 552 information required to populate all of its attributes with known data. CPE dictionaries support
 553 this situation, but they **MUST NOT** include such a name if a more complete version of that name
 554 exists in the dictionary.

⁴ Pseudocode that implements this rule is found in Section 8.2, in the function `contains-restricted-characters`.

⁵ In CPE Name Matching, the **SUPERSET** is non-proper, meaning that it includes cases where the two names are the same.

555 2. Existing name too general (the new name is a SUBSET of the old name, but the new name is not
 556 EQUAL to the old name). The new name may include information that was not available when
 557 previous, more general names were created. For example, there may already be a similar
 558 identifier name in the dictionary that contains the ANY logical value for fields that have known
 559 data in the new name.⁶ In such a case the pre-existing name(s) SHALL be deprecated to the new,
 560 more specific name; the deprecation process defined in Section 6.2 SHALL be followed for
 561 performing this deprecation.

562 To provide an example (focusing on the CPE update attribute), suppose that a CPE dictionary contains the
 563 following identifier name version of a CPE:

```
564 wfn: [part="a", vendor="foo_company", product="bar", version="2\.3",
565 update="sp1", edition=ANY, language=ANY, sw_edition=ANY, target_sw=ANY,
566 target_hw=ANY, other=ANY]
```

567 The above CPE represents the hypothetical product Foo Company Bar 2.3 sp1. At some later point in
 568 time, a dictionary contributor submits the following identifier name version of a CPE for potential
 569 inclusion within the same dictionary:

```
570 wfn: [part="a", vendor="foo_company", product="bar", version="2\.3",
571 update=ANY, edition=ANY, language=ANY, sw_edition=ANY, target_sw=ANY,
572 target_hw=ANY, other=ANY]
```

573 This new name represents the hypothetical product: Foo Company Bar 2.3. The difference with this new
 574 name is that "ANY" is the value for the update attribute. This means that the contributor of this name does
 575 not know the correct value of the update field at the time of submission. The acceptance criteria defined
 576 in this document does not permit this submission because an identifier name already exists that contains a
 577 known value for the update field of the product. The reason for this restriction is that the less-complete
 578 name does not represent any real-world product, but instead represents a set of existing identifier name
 579 within the dictionary. If the dictionary contributor discovers a new update of the product, then the
 580 contributor should submit an identifier name containing the new update value.

581 This does not mean that an identifier name representing the product Foo Company Bar 2.3 with no update
 582 is not permitted (e.g., the first release of the product contained no update). The identifier name
 583 representing the product with no update would contain the NA logical value in the update attribute. The
 584 following CPE does meet the acceptance criteria since it represents a real-world product that is known to
 585 have no update (i.e., the initial release of Bar 2.3):

```
586 wfn: [part="a", vendor="foo_company", product="bar", version="2\.3",
587 update=NA, edition=ANY, language=ANY, sw_edition=ANY, target_sw=ANY,
588 target_hw=ANY, other=ANY]
```

589 6.2 CPE Dictionary Deprecation Process

590 All identifier names stored within a dictionary MUST be immutable because data relating to a product is
 591 captured in the identifier itself. An identifier name MUST NOT be modified because the loss of historical
 592 information will cause problems for anyone already using that identifier name. Any updates to the product
 593 data captured in the identifier name MUST occur through deprecation. For example, when a dictionary

⁶ Pseudocode that implements this rule is found in Section 8.2, in the function matches-more-complete-in-dictionary.

594 maintainer discovers more information about a product represented by an existing identifier in the
 595 dictionary, they **MUST** deprecate the legacy identifier name in favor of a new identifier name or new set
 596 of identifier names. A deprecated identifier name is no longer considered valid.

597 An important distinction exists between the set of names that deprecate an identifier name and the
 598 identifier name that a CPE dictionary user selects as a replacement of the deprecated name. The CPE
 599 dictionary may link a deprecated identifier name to a set of identifier names replacing it, but this is
 600 asserting that any identifier name within the set of new identifier names is a valid replacement for the
 601 deprecated identifier name. Using this information a CPE dictionary user may, depending on the use case,
 602 decide on the appropriate name to use as a replacement, or decide to use the entire set. For example, when
 603 resolving the deprecating entries for a deprecated name found within an applicability statement, an
 604 organization may find it useful to use the entire set to avoid false negatives.

605 There are three types of identifier name deprecation:

- 606 ■ **Identifier Name Correction** – An error occurred during name creation. The error could range from a
 607 simple syntax error (e.g., typo, misspelling) to an incorrect product listing. For example, a dictionary
 608 maintainer may add a product to the dictionary, only to later discover that the product is only a
 609 component library of a larger product. In this case, the dictionary maintainer would deprecate the
 610 identifier name representing the component library to the identifier name representing the actual
 611 product containing the component. This type of deprecation is *one-to-one*, with the deprecated name
 612 pointing to the single identifier name that represents the correct product.
- 613 ■ **Identifier Name Removal** – An identifier name exists in the dictionary that does not belong and has
 614 no replacement. This condition normally results from a case where the dictionary maintainer has
 615 added a name to the dictionary that should never have been included in the dictionary. In this case,
 616 the dictionary maintainer will deprecate the legacy name without pointing to a new name.
- 617 ■ **Additional Information Discovery** – The dictionary maintainer adds one or more identifier names to
 618 the dictionary that are more complete than an existing identifier name. In this case, the pre-existing
 619 name really represents a set of possible products that the maintainer did not know about when
 620 originally adding the name to the dictionary. This type of deprecation may be *one-to-many* since the
 621 dictionary maintainer will deprecate the pre-existing name to all of the names that are more complete.
 622 This deprecation relationship is largely informational; when making this deprecation, the dictionary
 623 maintainer is asserting that any identifier name within the set of new identifier names is a valid
 624 replacement for the deprecated name. It is up to the user to decide on which individual name to use,
 625 or to use the entire set.

626 When deprecating an identifier name, the dictionary maintainer **MUST** link the deprecated identifier
 627 name to all identifier names that are replacing it through the deprecated-by relationship as defined in
 628 Section 5.6. For example, if x is a single identifier name that is being deprecated and y is a set of
 629 identifier names that is deprecating x, then "x deprecated-by y" defines a deprecation relationship
 630 between x and y.

631 When a dictionary maintainer decides to deprecate an identifier name, the dictionary maintainer **MUST**
 632 follow the below deprecation process. This process refers to the identifier name being deprecated as the
 633 *legacy name*.

- 634 1. If the dictionary maintainer is replacing the legacy name with one or more new identifier names,
 635 then the dictionary maintainer **MUST** add the new identifier names to the dictionary.

- 636 2. The dictionary maintainer **MUST** add a deprecation element to the legacy name dictionary entry
 637 to signify that it is deprecated. If the legacy name was previously deprecated, then the dictionary
 638 maintainer **MUST** add a new deprecation element to record the new deprecation.⁷
- 639 3. The dictionary maintainer **MUST** expand the new deprecation element to include one or more
 640 deprecated-by elements to record the identifier name(s) replacing the legacy name. The dictionary
 641 maintainer **MUST** specify the type of deprecation within the type attribute of each deprecated-by
 642 element. Additionally:
- 643 a. If the dictionary maintainer is performing an Identifier Name Correction deprecation, then the
 644 dictionary maintainer **MUST** populate the name attribute of the deprecated-by element with
 645 the identifier name replacing the legacy name.
- 646 b. If the dictionary maintainer is performing an Additional Information Discovery deprecation,
 647 then the dictionary maintainer **MUST** populate the name attribute of the deprecated-by
 648 element with a name representing the set of new identifier names that are replacing the legacy
 649 name. This name may contain wildcards (e.g., *, ?) to represent a set of identifier names.
- 650 4. The dictionary maintainer **MUST** record the change in the legacy name's provenance
 651 information.

652 **6.3 CPE Dictionary Provenance Data**

653 CPE dictionary maintainers **MUST** capture the required provenance data specified in Section 5.5. The
 654 provenance data required includes data useful in understanding the reasoning behind changes made to
 655 identifier names stored within a CPE Dictionary. This provenance data also captures the organization
 656 responsible for identifier name submissions, as well as the authority behind the submissions.

657 **6.4 CPE Dictionary Management Documents**

658 Every CPE dictionary has a set of supporting management documentation associated with it. This
 659 documentation improves the transparency relating to dictionary acceptance criteria, dictionary content
 660 creation decisions and processes associated with the dictionary. Each CPE dictionary creator or
 661 maintainer **MUST** create or reference a series of accompanying management documents capturing rules
 662 and processes specific to its dictionary. A single dictionary creator or maintainer **MAY** establish multiple
 663 extended CPE dictionaries. In these situations a single set of external CPE dictionary management
 664 documents **MAY** be referenced as the authoritative documents for each extended dictionary.

665 This section defines each required CPE dictionary management document.

666 **6.4.1 Dictionary Content Management and Decisions Document**

667 Dictionary maintainers **MUST** either create or reference a Dictionary Content Management and Decisions
 668 Document for each dictionary that they maintain. The document defines the procedures for identifier
 669 name creation within a particular dictionary or set of dictionaries. All procedures defined in the decisions
 670 document **MUST NOT** override or conflict with the requirements defined in this specification. The
 671 decisions document **MUST** capture the following information:

⁷ Multiple deprecations may occur against a single identifier WFN at different times (e.g., it is discovered that more products are included in the deprecated-by set). To support this, the CPE Dictionary data model requires that each deprecation element within an identifier WFN record the deprecation for one instant in time.

- 672
673
1. Rules relating to dictionary-specific acceptance criteria for identifier names.
- 674
675
676
677
678
2. Identification strategies relating to different product types. Due to the heterogeneous nature of product versioning in the IT industry, multiple disparate strategies for versioning products exist. Where possible, the document should record the different identification strategies captured within the dictionary. For example, if a specific product line uses seven digits within its version syntax, then the document should record the semantics of each digit within this syntax.
- 679
680
681
3. Automated identifier name creation strategies for specific products. These strategies may include the API calls or functions to call on certain products to populate specific attributes of an identifier name.
- 682
683
684
685
4. Lists of valid values for specific CPE attributes where appropriate. Valid value lists either may be global or pertain to specific CPEs. For example, it is possible to have a granular valid value list for the version attribute of a specific vendor and product, without extending the scope of this valid values list to every CPE version attribute within the dictionary.
- 686
687
5. Abbreviation rules for data within specific CPE attributes where appropriate. Abbreviation rules may either be global or pertain to specific CPEs.
- 688
6. Rules relating to any dictionary-specific provenance data that the dictionary records.

689 6.4.2 Dictionary Process Management Document

690 Dictionary maintainers **MUST** either create or reference a Dictionary Process Management Document for
691 each dictionary that they maintain. The document defines all processes associated with a particular
692 dictionary or set of dictionaries. All processes defined in the document **MUST NOT** override or conflict
693 with the requirements defined in this specification. The process management document **MUST** capture
694 the following information:

- 695
1. The scope of the dictionary.
- 696
2. The target audience of the dictionary.
- 697
698
699
700
3. The submission process that users must follow to submit new CPE identifiers for inclusion within the dictionary. At a minimum, this overview **SHOULD** include the submission format, the process for starting the submission process, and the workflow surrounding the submission process.
- 701
702
4. The content decisions process that the community follows to create the content decision rules that are captured in the Dictionary Content Management and Decisions Document.
- 703
5. The CPE Identifier modification process followed by dictionary maintainers.
- 704
705
6. The dictionary distribution process and methodology. At a minimum, this **SHOULD** define the binding in which the dictionary is distributed.

706

707 7. Dictionary Use

708 This section defines two common ways of using a CPE Dictionary. The first involves looking up a single
 709 identifier name within a dictionary. The second involves using a name representing a set of products to
 710 search against the dictionary to determine what identifier names within the dictionary are members of that
 711 set. The two methods of searching are necessary to support the type of information a user is retrieving
 712 from a dictionary. In some cases, it is necessary to retrieve an exact match from the dictionary; for
 713 example, when resolving a deprecation chain based on “Identifier Name Correction” deprecation, it is
 714 necessary to resolve only the exact identifier name listed in the deprecation metadata. In other cases, it is
 715 necessary to search for an entire set of CPE names; for example, when resolving a deprecation chain
 716 based on “Additional Information Discovery” deprecation, it is necessary to resolve all identifier names
 717 captured in the deprecation metadata.

718
 719 The section also explains how identifier names should be resolved from a deprecated name. All of these
 720 operations leverage the CPE name matching algorithms⁸ when performing the search operations.

721 7.1 Identifier Lookup

722 Identifier lookup involves using the WFN corresponding to a single identifier name (i.e., the source
 723 identifier name) to iterate through each entry in a dictionary to see if that source identifier exists within
 724 the dictionary—if any identifier name in the dictionary is equal to the source identifier name. Equality
 725 means that the corresponding attribute values in the two identifier names are all equal. The `CPE_EQUAL`
 726 function defined in the CPE Name Matching specification [CPE23-M:7.2] presents a formal
 727 implementation of this equality test. The two possible outcomes of an identifier lookup are:

- 728 ■ **Match** – The source identifier name matches an identifier name within the dictionary. The
 729 operation returns the status “EXACT-MATCH” and the WFN of the dictionary identifier name
 730 that matched. Note that since each name within the dictionary **MUST** be unique, at most only one
 731 identifier name could match the source identifier name.
- 732 ■ **No Match** – The source identifier name does not match any identifier name within the dictionary.
 733 The operation returns a null value.

734
 735 Section 8.3 defines the formal implementation of this identifier lookup operation in pseudocode.

736 7.2 Dictionary Searching

737 Dictionary searching involves using a source name representing a set of products to search against a
 738 dictionary to determine which identifier names within the dictionary are members of that set. A dictionary
 739 searching routine would iterate through each entry in the dictionary to identify all that are a member of
 740 the set represented by the source name. Possible outcomes of this searching operation are:

- 741 ■ **Superset Match** – One or more identifier names from the dictionary belong to the set represented
 742 by the source name. In set theory language, the set represented by the source name is a superset of
 743 the matched identifier names from the dictionary. The operation returns “SUPERSET-MATCH”
 744 and the set of matching dictionary identifier names expressed as WFNs.
- 745 ■ **Subset Match** – The set represented by the source name is a possible subset of one or more
 746 identifier names within the dictionary. This will only occur if the source name is more specific

⁸ The CPE Name Matching specification [CPE23-M] formally defines the CPE name matching algorithms used in this section, and in accompanying pseudocode implementations in Section 8.3.

747 than one or more names within the dictionary. Generally this result should be treated as an error;
 748 the searcher may want to notify the dictionary maintainer that a name in the wild is more specific
 749 than one or more identifier names within the dictionary. The operation returns “SUBSET-
 750 MATCH” and the set of matching dictionary identifier names expressed as WFNs.

751 ■ **No Match** – There is no relationship between the set represented by the source name and the
 752 identifier names within the dictionary. The source name is disjoint with the dictionary. When no
 753 match is found, the operation returns a null value.

754
 755 Section 8.3 defines the formal implementation of this dictionary search operation in pseudocode. This
 756 pseudocode leverages the `CPE_SUPERSET` and `CPE_SUBSET` functions defined in the CPE Name
 757 Matching specification [CPE23-M:7.2] to implement the search operation.

758 7.3 Use of Deprecated Identifier Names

759 The concept of deprecated identifier names is explained in Section 6.2. CPE dictionary users **MUST NOT**
 760 use a deprecated identifier name, but **MUST** instead use an identifier name that is linked to the deprecated
 761 identifier name through the deprecated-by relationship.⁹ It is important to understand that even though a
 762 set of names may deprecate one identifier name, an organization does not have to use all of these new
 763 names. The organization **MAY** simply pick a non-deprecated name out of the set, or the organization
 764 **MAY** choose to use the entire set.

765 When a product consumes or outputs an identifier name, that product **MUST** first determine if the
 766 identifier name is deprecated in the Official CPE Dictionary. If the identifier name is not present in the
 767 Official CPE Dictionary, then the product **MUST** determine if the identifier name is deprecated in an
 768 extended CPE dictionary to which it has access. If the identifier name is deprecated, the product **MUST**
 769 resolve the correct non-deprecated identifier names for use in place of the deprecated identifier name. The
 770 product **MUST** use the following process, which references the `dictionary-search` function
 771 defined in Section 8.3, to perform this resolution.

- 772 1. If the deprecated identifier name does not reference any identifier names within the *deprecated-*
 773 *by* element, then the organization **MUST** choose a new non-deprecated identifier name from the
 774 dictionary. This situation will occur if the deprecation is of type “Identifier Name Removal”.
- 775 2. If the deprecated identifier name provides an identifier name within the *deprecated-by* element,
 776 and:
 - 777 a. If the deprecation type is “Identifier Name Correction”, then the organization **MUST** resolve
 778 the dictionary entry containing the identifier name listed. The organization **MUST** resolve
 779 this identifier name using the `dictionary-search` function, passing the identifier name,
 780 the dictionary, and a value of *true* as the parameters to the function; these arguments will
 781 result in an identifier lookup operation.
 - 782 b. If the deprecation type is “Additional Information Discovery” then the organization **MUST**
 783 resolve the set of dictionary entries containing the identifier names. The organization **MUST**
 784 resolve this set of identifier names using the `dictionary-search` function, passing the
 785 set of identifier names, the dictionary, and a value of *false* as the parameters to the function;
 786 these arguments will result in a dictionary search operation.

⁹ This requirement **MAY** be ignored when dictionary users are purposefully intending to communicate information relating to deprecated WFNs.

- 787
788
789
790
3. If there are multiple deprecation elements or a single deprecation element with multiple deprecated-by elements, then the organization **MUST** iterate through step 2 for all deprecated-by elements provided. The organization **MUST** take the union of all resolved sets of identifier names; this union is the correct set of identifier names that have replaced the legacy name.
- 791
792
793
4. The organization **MAY** encounter deprecated identifier names in the set of identifier names resolved in Step 2. In this case, the organization **MUST** follow the above process to replace these deprecated names with the set of names that deprecated it; this process may be recursive.
- 794
795
796
5. The final set of resolved names represents all names that replace the legacy name. Organizations **MAY** either use all of these names, or pick one name out of the set to use in place of the legacy name.

DRAFT

797 **8. CPE Dictionary Operations and Pseudocode**

798 This section specifies functions for basic CPE dictionary operations and provides pseudocode for the
799 identifier name acceptance criteria and the dictionary searching operations.

800 **8.1 Operations on a CPE Dictionary**

801 This section defines a set of functions for performing common activities against a CPE Dictionary. These
802 functions are relatively simple and do not require a pseudocode implementation, but the remaining sub-
803 sections present pseudocode that may utilize these common functions.

804 **8.1.1 Function `get_cpe_items(d)`**

805 The `get_cpe_items(d)` function takes a single CPE Dictionary, `d`, and returns all `cpe-items`
806 associated with it. A single `cpe-item` within the dictionary represents the identifier name and all
807 associated metadata.

808 **8.1.2 Function `get_cpe_item_WFN(item)`**

809 The `get_cpe_item_WFN(item)` function takes a single `cpe-item` element and returns the
810 identifier WFN that it represents. Because the dictionary holds bound representations of names, it is
811 necessary to unbind one of those to obtain a WFN. The formatted string binding should be used if
812 available, and it can be unbound using the `unbind_fs(fs)` function defined in the CPE Naming
813 Specification [CPE23-N:6.2.3].

814 **8.1.3 Function `get(w,a)`**

815 The `get(w, a)` function takes two arguments, a WFN `w` and an attribute `a`, and returns the value of `a`. If
816 the attribute `a` is unspecified in `w`, the function returns the default value ANY. This function is defined in
817 the CPE Naming Specification [CPE23-N:5.4.2].

818 **8.1.4 Function `is_deprecated(item)`**

819 The `is_deprecated(item)` function takes a single `cpe-item` and returns `true` if the `cpe-item`
820 is deprecated, `false` if the `cpe-item` is not deprecated.

821 **8.1.5 Function `getItem(list, index)`**

822 The `getItem(list, index)` function is a helper function for retrieving an item in a list. The
823 function will return the list item at the position specified by `index`. This function assumes a 0-based
824 index and will return `null` if no items exist at the provided index.

825 **8.1.6 Function `strlen(s)`**

826 The `strlen(s)` function returns the length of string `s`. If the string is empty, it returns zero. It is defined
827 as in GNU C.

828 8.1.7 **Function substr(s,b,e)**

829 The `substr(s,b,e)` function returns a substring of string `s`, beginning with the character at location `b`
 830 (with zero being the first character) and ending with the character at location `e`. If `b` equals `e`, it returns
 831 the character at location `b`. `b` must be less than or equal to `e`. If `b` is equal to or greater than `strlen(s)`,
 832 it returns `nil`.

833 **8.2 Acceptance Criteria Pseudocode**

834 This section defines the algorithm required to implement the acceptance criteria defined in Section 6.1.
 835 The core algorithm is implemented in the below pseudocode function named `accept-name`, which
 836 processes a given formatted string binding, `fs`, converts it to its WFN representation, `w`, and compares
 837 the WFN against a specific dictionary to determine if the dictionary should accept the new name. The
 838 following list provides a brief summary of the algorithm implemented in the `accept-name` function:

- 839 1. Unbind the formatted string `fs` to its WFN representation `w` because all the other functions within
 840 `accept-name` require a WFN as input. See the `unbind_fs` function in the CPE Naming
 841 Specification [CPE23-N:6.2.3].
- 842 2. Use the helper function `contains-restricted-characters` to determine if the WFN `w`
 843 contains any of the restricted characters defined in Section 6.1.1.3. If `w` contains any of the
 844 restricted characters, `accept-name` will return `false`.
- 845 3. Use the helper function `contains-required-attributes` to determine if the WFN `w`
 846 contains known data for all of the required attributes specified in Section 6.1.2. If `w` does not
 847 contain all of the required attributes, `accept-name` will return `false`.
- 848 4. Use the helper function `matches-more-complete-in-dictionary` to determine if the
 849 WFN `w` is unique within the dictionary `d`, as specified in Section 6.1.3. If `w` does not adhere to
 850 this rule, or in other words if `w` matches against a more complete name in the given dictionary `d`,
 851 then `accept-name` will return `false`. This function uses the `dictionary-search` function
 852 defined in Section 8.3 for matching against the dictionary.

```

853
854 1 function accept-name(fs, d)
855 2   ;; Top-level function to determine if the formatted string fs
856 3   ;; should be accepted into dictionary d based on high-level
857 4   ;; acceptance criteria. Assumes that the fs and its WFN
858 5   ;; representation meet acceptance criteria defined in the
859 6   ;; CPE Naming Spec.
860 7   w =: unbind_fs(fs).
861 8   if contains-restricted-characters(w)
862 9     then return false.
863 10  endif.
864 11  if !contains-required-attributes(w)
865 12    then return false.
866 13  endif.
867 14  if matches-more-complete-in-dictionary(w, d)
868 15    then return false.
```

```

869 16  endif.
870 17  return true.
871 18  end.

```

872 **Figure 8-1: accept-name function**

```

873
874 1  function contains-restricted-characters(w)
875 2  ;; Helper-function to determine if WFN w contains characters
876 3  ;; not permitted in dictionary.
877 4  ;; loop through every attribute in WFN
878 5  foreach a in {part,vendor,product,version,update,edition,language,
879 6  sw_edition,target_sw,target_hw,other} do
880 7  s := get(w,a). ;; get value of attribute
881 8  if (s = ANY or s = NA)
882 9  ;; the attribute value is a logical
883 10 then continue. ; to next attribute
884 11 endif.
885 12 ;; If we get here, s is a string value.
886 13 n := 0.
887 14 loop
888 15 if n >= strlen(s)
889 16 then break. ;; break to outer loop
890 17 endif.
891 18 c := substr(s,n,n). ;; get the n'th character of s.
892 19 if (c = "*" or c = "?")
893 20 then
894 21 if ((n = 0) or (substr(s,n-1,n-1) != "\"))
895 22 then return true. ;; unquoted '*' or '?' not permitted
896 23 endif.
897 24 else
898 25 ;; character is legal, move on
899 26 n := n + 1.
900 27 continue.
901 28 endif.
902 29 endloop.
903 30 endfor.
904 31 return false.
905 32 end.

```

906 **Figure 8-2: contains-restricted-characters function**

```

907
908 1  function contains-required-attributes(w)
909 2  ;; Helper-function to determine if required attributes contain
910 3  ;; known data. WFN syntax defined in CPE Naming Spec ensures all
911 4  ;; attributes contain at least some data.
912 5  foreach a in {part, vendor, product, version} do
913 6  s := get(w,a). ;; get string value of attribute
914 7  ;; only loop through required attributes of w
915 8  if s = ANY
916 9  then return false.
917 10 endif.

```

```

918 11     if (a != version and s = NA)
919 12         then return false. ;; NA only permitted in version
920 13     endif.
921 14 endfor.
922 15 return true.
923 16 end.

```

Figure 8-3: contains-required-attributes function

```

925
926 1 function matches-more-complete-in-dictionary(w, d)
927 2     ;; Helper-function to determine if identifier WFN w matches a
928 3     ;; more complete name in the dictionary d (i.e. a superset match).
929 4     matches := dictionary-search(w, d, false).
930 5     if (size(matches) > 0)
931 6         then
932 7             if (getItem(matches, 0) = "SUPERSET-MATCH")
933 8                 then return false. ;; at least one superset match was found
934 9             endif.
935 10        endif.
936 11        return true. ;; no match, or subset match are both okay.
937 12    end.

```

Figure 8-4: matches-more-complete-in-dictionary function

939 8.3 Dictionary Use Pseudocode

940 This section contains pseudocode that implements the identifier lookup and dictionary search operations
 941 defined in Section 7 with the *dictionary-search* function.¹⁰ This function is focused on returning
 942 all non-deprecated names within a dictionary, but will not filter out deprecated names from the result set,
 943 allowing the caller to filter the names based on use case. While this function will not filter out deprecated
 944 names, it does not guarantee that it will return all matching deprecated names in the result set; it only
 945 guarantees the return of all matching non-deprecated names. When not looking for exact matches, the
 946 function will look for all superset matches within the given dictionary. The function will only look for
 947 subset matches if the given dictionary contains no superset matches. This ordering derives from the CPE
 948 Dictionary acceptance criteria that will not allow a dictionary to contain non-deprecated subset matches if
 949 a superset match is present.¹¹

950 Table 8-1 provides a detailed overview of the *dictionary-search* function. This pseudocode
 951 leverages the accessor functions defined in Section 8.1. The code also leverages the CPE_EQUAL,
 952 CPE_SUBSET, and CPE_SUPERSET functions defined in the CPE Name Matching specification
 953 [CPE23-M:7.2].

954

¹⁰ This function does not account for deprecation chains. For example, if an identifier WFN is found that is deprecated, a tool may want to recursively search all the identifier WFNs in the deprecation chain (i.e., the set of all identifier WFNs referenced by the deprecated-by property).

¹¹ The *dictionary-search* function was designed for readability; more efficient methods for implementing this logic exist.

955

Table 8-1: Description of dictionary-search function

Line Number(s)	Description
1	The function accepts three arguments, the source WFN, the dictionary to search against, and a boolean flag 'exact'. When 'exact' is true the function will perform an identifier lookup based on the source WFN. When 'exact' is false the function will perform dictionary searching for all names contained the source WFN set.
11	Creates a list to store all discovered matches as well as the match type; if a match is found, then the first position of this list will contain the match type and the second position will contain the matches (either single match, or set of matches).
12 -13	If 'exact' is true, the function will enter into logic to determine if an exact match is present within d. The function will then call the findExactMatch function (defined in Figure 8-8) to determine if an exact match exists in the dictionary.
14-20	If an exact match is found, the function will populate the 'response' list with the response type of 'EXACT-MATCH' and then append the exact match to the next position in the list. The function will then return the response list, or null if no match was found.
22	The function will call the findSupersetMatches function (defined in Figure 8-6) to build the set of all superset matches found in d.
23-28	If any superset matches are found, the function will populate the 'response' list with the response type of 'SUPERSET-MATCH' and then append the set of matches to the next position in the list. The function will then return the response list.
29	The function will call the findSubsetMatches function (defined in Figure 8-7) to build the set of all subset matches found in d. NOTE: No non-deprecated subset matches should be present if a superset match was found.
30-35	If any subset matches are found, the function will populate the 'response' list with the response type of 'SUBSET-MATCH' and then append the set of matches to the next position in the list. The function will then return the response list.
36	If no matches were found the function will return null.

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

```

1  function dictionary-search(source, d, exact)
2      ;; For a given source WFN, source, the function will determine
3      ;; how it relates to the given dictionary, d. If the source WFN
4      ;; is a superset match to one or more identifier WFNs in d, then
5      ;; the function will return that set of names. If the source name
6      ;; is not a superset of any dictionary names, but is a subset of a
7      ;; dictionary name(s), then the function will return the set of
8      ;; dictionary names of which it is a subset. If an exact match is
9      ;; found and exact is true, the function will return the exact
10     ;; match. If no match exists the function will return null.
11     response := new List().
12     if (exact = true)
13         match := findExactMatch(source, d).
14         if (match != null)
15             then
16                 response := append(response, "EXACT-MATCH").
17                 response := append(response, match).
18                 return response.
19             else return null.
20         endif.
21     endif.
22     supersetMatches := findSupersetMatches(source, d).
23     if (size(supersetMatches) > 0)

```

```

981 24     then ;;superset matches found
982 25     response := append(response, "SUPERSET-MATCH").
983 26     response := append(response, supersetMatches).
984 27     return response.
985 28 endif.
986 29 subsetMatches := findSubsetMatches(source, d).
987 30 if (size(subsetMatches) > 0)
988 31     then ;; subset matches found
989 32     response := append(response, "SUBSET-MATCH").
990 33     response := append(response, subsetMatches).
991 34     return response.
992 35 endif.
993 36 return null.
994 37 end

```

Figure 8-5: dictionary-search function

Table 8-2: Description of findSupersetMatches function

Line Number(s)	Description
4	Creates a set to store all discovered superset matches.
5	Starts looping over every cpe-item in given dictionary, d.
7	Retrieves the WFN represented by the current item from the dictionary.
8-12	Passes the source and dictionary WFN to the CPE_SUPERSET and CPE_EQUAL functions (defined in the CPE Name Matching specification [CPE23-M:7.2] to determine how the two names relate. If the source is a 'superset' of the dictionary name, but not equal to the dictionary name, then the function will append the item to the set of matches.
14	The function will return the set of matches.

```

997
998
999 1 function findSupersetMatches(source, d)
1000 2 ;; For a given source WFN, source, the function will find all
1001 3 ;; superset matches contained within the given CPE dictionary, d.
1002 4 matches := new Set().
1003 5 foreach item in get_cpe_items(d)
1004 6     do
1005 7         dictionaryName := get_cpe_item_WFN(item). ;;WFN from cpe-item
1006 8         if ((CPE_SUPERSET(source, dictionaryName) = true) and
1007 9             (CPE_EQUAL(source, dictionaryName) = false))
1008 10             then
1009 11                 matches := append(matches, item).
1010 12             endif.
1011 13         endfor.
1012 14 return matches.
1013 15 end

```

Figure 8-6: findSupersetMatches function

1016

Table 8-3: Description of findSubsetMatches function

Line Number(s)	Description
4	Creates a set to store all discovered subset matches.
5	Starts looping over every cpe-item in given dictionary, d.
7	Retrieves the WFN represented by the current item from the dictionary.
8-12	Passes the source and dictionary WFN to the CPE_SUBSET and CPE_EQUAL functions (defined in the CPE Name Matching specification [CPE23-M:7.2] to determine how the two names relate. If the source is a 'subset' of the dictionary name, but not equal to the dictionary name, then the function will append the item to the set of matches.
14	The function will return the set of matches

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

```

1  function findSubsetMatches(source, d)
2  ;; For a given source WFN, source, the function will find all
3  ;; subset matches contained within the given CPE dictionary, d.
4  matches := new Set().
5  foreach item in get_cpe_items(d)
6    do
7      dictionaryName := get_cpe_item_WFN(item). ;;WFN from cpe-item
8      if ((CPE_SUBSET(source, dictionaryName) = true) and
9          (CPE_EQUAL(source, dictionaryName) = false))
10     then
11       matches := append(matches, item).
12     endif.
13 endfor.
14 return matches.
15 end

```

1033

Figure 8-7: findSubsetMatches function

1034

Table 8-4: Description of findExactMatch function

Line Number(s)	Description
4	Starts looping over every cpe-item in given dictionary, d.
6	Retrieves the WFN represented by the current item from the dictionary.
7-10	Passes the source and dictionary WFN to the CPE_EQUAL function (defined in the CPE Name Matching specification [CPE23-M:7.2] to determine how the two names relate. If the two names are equal then the function will return the item as the exact match.
12	If not exact matches were found the function will return null.

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

```

1  function findExactMatch(source, d)
2  ;; For a given source WFN, source, the function will find the
3  ;; exact match contained within the given CPE dictionary, d.
4  foreach item in get_cpe_items(d)
5    do
6      dictionaryName := get_cpe_item_WFN(item). ;;WFN from cpe-item
7      if (CPE_EQUAL(source, dictionaryName) = true)
8      then

```

```
1045 | 9         return item.  
1046 | 10        endif.  
1047 | 11    endfor.  
1048 | 12    return null.  
1049 | 13 end
```

Figure 8-8: findExactMatch function

1050

DRAFT

1051 Appendix A—References

1052 The following documents are indispensable references for understanding the application of this
1053 specification.

1054

1055 Normative References

1056

1057 [CPE22] Buttner, A. and N. Ziring, *Common Platform Enumeration (CPE)—Specification, Version 2.2*,
1058 March 11, 2009. See http://cpe.mitre.org/specification/archive/version2.2/cpe-specification_2.2.pdf.

1059

1060 [CPE23-M] Parmelee, M., Booth, H., Waltermire, D., and Scarfone, K., NIST Interagency Report 7696,
1061 *Common Platform Enumeration: Name Matching Specification Version 2.3 (Draft)*, April 2011. See
1062 <http://csrc.nist.gov/publications/PubsDrafts.html>.

1063

1064 [CPE23-N] Cheikes, B., Waltermire, D., and Scarfone, K., NIST Interagency Report 7695, *Common*
1065 *Platform Enumeration: Naming Specification Version 2.3 (Draft)*, April 2011. See
1066 <http://csrc.nist.gov/publications/PubsDrafts.html>.

1067

1068 Informative References

1069

1070 [SP800-117] Quinn, S., Scarfone, K., Barrett, M., and Johnson, C., NIST Special Publication 800-117,
1071 *Guide to Adopting and Using the Security Content Automation Protocol*, July 2010. See:
1072 <http://csrc.nist.gov/publications/nistpubs/800-117/sp800-117.pdf>.

1073

1074 [SP800-126] Waltermire, D., Quinn, S., and Scarfone, K., NIST Special Publication 800-126 Revision 1,
1075 *The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.1*,
1076 February 2011. See <http://csrc.nist.gov/publications/nistpubs/800-126-rev1/SP800-126r1.pdf>.

1077

1078 **Appendix B—Change Log**

1079 **Release 0 – 9 June 2010**

- 1080
- Complete draft specification released to the CPE community for comment.

1081

1082 **Release 1 – 30 June 2010**

- 1083
- Minor edits to audience description.
 - Minor editorial changes throughout the document.
 - Removed all mention of and support for the logical value UNKNOWN.
 - Updated Dictionary Searching section to remove the notion of an Error result, and clarify on superset versus subset matches.
 - Updated deprecation logic, and data model to include three distinct types of deprecation.

1089

1090 **Release 2 – 2 June 2011**

- 1091
- Reorganized the sequence of several sections and sub-sections.
 - Minor editorial changes throughout the document.
 - Reorganized the data model discuss to include all elements and attributes available in version 2.3.
 - Changed the overarching term “Dictionary Searching” to “Dictionary Use” to distinguish it from the lower-level term “dictionary searching”.
 - Clarified what results are returned by identifier lookups and dictionary searching.
 - Made several minor changes to the version 2.3 schema/data model, including making several elements and attributes optional and changing a few types.

1099