NIST Interagency Report 7695

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

Common Platform Enumeration: Naming Specification Version 2.3

Brant A. Cheikes
David Waltermire
Karen Scarfone

NIST Interagency Report 7695

Common Platform Enumeration: Naming Specification Version 2.3

Brant A. Cheikes
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

August 2011

U.S. Department of Commerce

Rebecca M. Blank, Acting Secretary

National Institute of Standards and Technology

Patrick D. Gallagher, Under Secretary for Standards and Technology and 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 7695
49 pages (Aug. 2011)

Acknowledgments

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

Abstract

This report defines the Common Platform Enumeration (CPE) Naming version 2.3 specification. The CPE Naming specification is a part of a stack of CPE specifications that support a variety of use cases relating to IT product description and naming. The CPE Naming specification defines the logical structure of names for IT product classes and the procedures for binding and unbinding these names to and from machine-readable encodings. This report also defines and explains the requirements that IT products must meet for conformance with the CPE Naming version 2.3 specification.

Trademark Information

Table of Contents

List of Figures and Tables

1. Introduction

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

CPE consists of several modular specifications. Combinations of the specifications work together in layers to perform various functions. This specification, CPE Naming, defines standardized methods for assigning names to IT product classes. An example is the following name representing Microsoft Internet Explorer 8.0.6001 Beta:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update="beta"]
```

This method of naming is known as a well-formed CPE name (WFN). It is an abstract logical construction. The CPE Naming specification defines procedures for binding WFNs to machine-readable encodings, as well as unbinding those encodings back to WFNs. One of the bindings, called a Uniform Resource Identifier (URI) binding, is included in CPE version 2.3 for backward compatibility with CPE version 2.2 [CPE22]. The URI binding representation of the WFN above is:

```
cpe:/a:microsoft:internet_explorer:8.0.6001:beta
```

The second binding defined in CPE 2.3 is called a formatted string binding. It has a somewhat different syntax than the URI binding, and it also supports additional product attributes. With the formatted string binding, the WFN above can be represented by the following.

```
cpe:2.3:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:*
```

The WFN concept and the bindings defined by the CPE Naming specification are the fundamental building blocks at the core of all CPE functionality.

1.1 Purpose and Scope

This report defines the specification for CPE Naming version 2.3, including the requirements for WFNs and name bindings. This report also defines and explains the requirements that CPE Naming implementations, such as software and services, must meet to claim conformance with the CPE Naming version 2.3 specification.

This report only applies to version 2.3 of CPE Naming. All other versions are out of the scope of this report, as are all CPE specifications other than CPE Naming.

1.2 Document Structure

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

- Section 2 defines selected terms and abbreviations used in this specification.
- Section 3 provides an overview of related specifications and standards.

- Section 4 defines the high-level conformance rules for this specification.
- Section 5 defines the WFN data model.
- Section 6 defines the procedures for binding and unbinding WFNs into and out of the URI and formatted string bindings.
- Section 7 defines the procedures for converting between bound forms.
- Appendix A lists normative and informative references.
- Appendix B provides a change log that documents significant changes to major drafts of the specification.

1.3 Document Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

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

Normative references are listed in Appendix A. The following reference citation conventions are used in the text of this document:

- For normative references, a square bracket notation containing an abbreviation of the overall reference citation, followed by a colon and subsection citation where applicable (e.g. [CPE22:7] is a citation for the CPE 2.2 Naming specification, Section 7);
- For references within this document (internal references) and non-normative references, a parenthetical notation containing the “cf.” (compare) abbreviation followed by a section number for internal references or an external reference, (e.g. (cf. 5.3.1) is a citation for Section 5.3.1 of this document).

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

When reading pseudocode the following should be kept in mind:

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

2. Definitions and Abbreviations

This section defines a set of common terms used within the document. Many terms have been imported from Section 4 of [ISO19770-2]. These are indicated by appending the particular subsection citation to the overall reference citation separated by a colon, e.g., [ISO19770-2:4.1.1]. This section also defines selected abbreviations, including acronyms, used within the document.

2.1 Definitions

Application: A system for collecting, saving, processing, and presenting data by means of a computer [ISO19770-2:4.1.1]. The term *application* is generally used when referring to a component of software that can be executed. The terms *application* and *software application* are often used synonymously.

Bind: To deterministically transform a logical construct into a machine-readable representation suitable for machine interchange and processing. The result of this transformation is called a *binding*. A binding may also be referred to as the "bound form" of its associated logical construct.

Component: An entity with discrete structure, such as an assembly or software module, within a system considered at a particular level of analysis [ISO19770-2:4.1.3]. Component refers to a part of a whole, such as a component of a software product, a component of a software identification tag, etc.

Computing Device: A functional unit that can perform substantial computations, including numerous arithmetic operations and logic operations without human intervention [ISO19770-2:4.1.4]. A computing device can consist of a standalone unit or several interconnected units. It can also be a device that provides a specific set of functions, such as a phone or a personal organizer, or more general functions such as a laptop or desktop computer.

Configuration Item: An item or aggregation of hardware or software or both that is designed to be managed as a single entity [ISO19770-2:4.1.5]. Configuration items may vary widely in complexity, size and type, ranging from an entire system including all hardware, software and documentation, to a single module, a minor hardware component or a single software package.

Hardware Device: A discrete physical component of an information technology system or infrastructure. A hardware device may or may not be a computing device (e.g., a network hub, a webcam, a keyboard, a mouse).

Operating System: A computer program, implemented in either software or firmware, which acts as an intermediary between users of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute applications [Silberschatz].

Platform: A computer or hardware device and/or associated operating system, or a virtual environment, on which software can be installed or run [ISO19770-2:4.1.17]. Examples of platforms include Linux™, Microsoft Windows Vista®, and Java™.

Product: A complete set of computer programs, procedures and associated documentation and data designed for delivery to a software consumer [ISO19770-2:4.1.19].

Release: A collection of new and/or changed configuration items which are tested and introduced into a production environment together [ISO19770-2:4.1.21].

Software: All or part of the programs, procedures, rules, and associated documentation of an information processing system [ISO19770-2:4.1.25].

Unbind: To deterministically transform a binding into its logical-form construct.

Uniform Resource Identifier (URI): A compact sequence of characters that identifies an abstract or physical resource available on the Internet. The syntax used for URIs is defined in [RFC3986].

2.2 Abbreviations

ABNF	Augmented Backus-Naur Form
ASCII	American Standard Code for Information Interchange
CPE	Common Platform Enumeration
IEC	International Electrotechnical Commission
IR	Interagency Report
ISO	International Organization for Standardization
IT	Information Technology
ITL	Information Technology Laboratory
NIST	National Institute of Standards and Technology
RFC	Request for Comment
URI	Uniform Resource Identifier
UTF-8	Unicode Transformation Format-8
WFN	Well-Formed CPE Name
XML	Extensible Markup Language

3. Relationship to Existing Specifications and Standards

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

3.1 Other CPE Version 2.3 Specifications

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

The CPE Naming version 2.3 specification defines the lowest layer of the stack; all other CPE specifications build upon it.

3.2 CPE Version 2.2

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

There have been significant changes in naming from CPE version 2.2 to 2.3. [CPE22] defines the CPE name as a multi-component URI obeying a specified grammar. CPE Naming version 2.3 departs significantly from that practice by introducing the WFN construct and defining procedures for binding and unbinding this construct to and from machine-readable representations.

The principal motivation for this change was to create opportunities for future growth and innovation in the ways in which machines exchange product descriptions. During CPE version 2.3's development, a clear need was recognized to define at least two different machine-readable representations for product descriptions, one for backward compatibility with prior releases of the CPE specifications, and a second to provide critical new features demanded by the user community. As work advanced, community members proposed additional transport representations for consideration. As the inventory of potential representations increased, it became clear that there could be serious challenges involved in defining numerous conversions among transports and procedures for pair-wise comparison. Consequently, an abstract canonical form—a kind of interlingua—was chosen to serve as the standardized form for processing CPE information. Using this interlingua it is possible to define conversions simply in terms of transforms into and out of the canonical form, and to define matching and other higher-level processes in generic rather than representation-specific terms. The WFN form lays the foundation for new binding forms to be introduced in the future without affecting other specifications defined in terms of the canonical form.

3.3 ISO/IEC 19770-2

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) have published ISO/IEC-19770 Part 2, "Software Identification Tag". As explained in the introduction to the standard, "The software identification tag is an XML file containing authoritative identification and management information about a software product. The software identification tag is installed and managed on a computing device together with the software product." [ISO19770-2], p. vi.

Both CPE and ISO/IEC-19770-2 address the need to standardize the way products are identified. There are many areas in which the two efforts overlap or complement each other, but CPE differs in a number of respects:

- CPE's scope is broader; it includes hardware devices, and it distinguishes operating systems from general software applications.
- CPE emphasizes the development and use of "common identifiers" that enable a wide variety of information about the same product or class of products to be correlated.
- CPE provides support for the creation of product descriptions (names for sets of products) as well as product identifiers (names for individual products).

4. Conformance

Products may want to claim conformance with this specification for a variety of reasons. This section provides the high-level requirements that must be met by any implementation seeking to claim conformance with this specification. These requirements are intended to guarantee that a conformant implementation not only can produce and/or consume the newly-introduced formatted string binding form as needed to interoperate with other implementations, but also can process legacy product identifiers as well.

The following apply to all implementations claiming conformance with this specification:

1. An implementation MUST make an explicit claim of conformance to this specification in any documentation provided to end users.

2. If the implementation produces (i.e., generates as an output) CPE names, it MUST produce syntactically correct formatted string bindings as needed to describe or identify applications, operating systems, and hardware devices (cf. 6.2).

3. If the implementation produces CPE names in URI form, it MUST produce URIs that adhere to the syntax rules specified in Figure 6-1, and it SHOULD produce URIs that adhere to the more constrained rules specified in Figure 6-2.

4. If the implementation consumes (i.e., accepts as valid input) CPE names, (a) it MUST consume syntactically correct formatted string bindings as needed to describe or identify applications, operating systems, and hardware devices (cf. 6.2); (b) it SHOULD be able to consume any CPE name that meets the syntax requirements specified in Figure 6-1 or Figure 6-2; and (c) it SHOULD be able to convert input CPE names in URI form to syntactically correct formatted string bindings (cf. 7.1).

5. It is OPTIONAL for implementations to be able to convert any syntactically correct formatted string binding to a valid CPE name that meets the requirements specified in [CPE22] (cf. 7.2). An implementation that offers this functionality SHOULD implement the procedure defined in Section 7.2. This functionality may enable an implementation to interoperate to a limited extent with implementations conforming to [CPE22] and possibly prior releases as well.

5. Well-Formed CPE Name Data Model

This section defines the foundational logical construct of CPE—the well-formed CPE name (WFN). WFNs can be used to describe a set of products or to identify an individual product. *Describing* an entity involves enumerating a set of attributes and their values to distinguish that entity from other entities in a domain. For example, Joe might describe his car as a "2004 Subaru Outback with a black leather interior". Conceptually, this description could be modeled as a set of attribute=value pairs:

[year=2004, maker=subaru, model=outback, interior_color=black, interior_material=leather]

A description is said to be *ambiguous* for a domain when it cannot be used to distinguish a unique entity possessing all specified attributes and values. The above description is ambiguous for a broad domain, such as all automobiles registered in the state of Massachusetts, but might not be ambiguous for a narrower domain, such as all automobiles registered in a particular nine-digit United States postal code region. To *identify* an entity is to describe it uniquely regardless of the domain it is in.

WFNs are used to describe and identify product classes. An example of a product class is a Lenovo ThinkPad X61; this class could be used to identify a hardware device's class. More general product classes, such as Lenovo Thinkpads or Lenovo laptops, could be used as class descriptions, not class identifiers. The CPE Naming specification leaves all decisions about what distinguishes identifiers and descriptions to the users. All names constitute descriptions of product classes, and a description becomes an identifier when it contains sufficient information to select a single entity from the universe of possibilities.

WFNs are used solely for product classes. They cannot identify product instances, which are unique, physically discernible entities in the world, such as a specific licensed and configured installation of a product on a particular computing device owned by XYZ Corp. and physically installed in a particular location in the world. Other aspects of product naming and description that are outside the scope of the CPE Naming specification are:

- Representing relationships (e.g., part of, bundled with, released before/after, same as) between products
- Representing user-defined configurations of installed products
- Representing entitlement/licensing information about products
- Defining procedures and guidelines for assigning "correct" or "valid" values to attributes of product descriptions or identifiers
- Defining procedures and guidelines for creating or maintaining valid-values lists.

5.1 Definitions and Notation

A *WFN* is an unordered set of attribute-value pairs that collectively (a) describe or identify a software application, operating system, or hardware device, and (b) satisfy the criteria specified in Section 5.2. *Unordered* means that there is no prescribed order for the attribute-value pairs, and there is no specified relationship (hierarchical, set-theoretic, or otherwise) among attributes or attribute-value pairs.

The WFN is a logical construct only. The WFN is not intended to be a data format, encoding, or any other kind of machine-readable representation for machine interchange and processing. Rather, it is a conceptual data structure—an abstract canonical form—used to clearly and unambiguously specify desired implementations and behaviors. There is no requirement that CPE-conformant tools create or manipulate WFN-like data structures internally to their implementations. Section 6 describes procedures for *binding* WFNs to machine-readable representations for interchange and processing.

An *attribute-value pair* is a tuple *a=v* in which *a* (the *attribute*) is an alphanumeric label (used to represent a property or state of some entity), and *v* (the *value*) is the value assigned to the attribute. Lexical case SHALL NOT distinguish attributes from one another, e.g., the attributes Foo, foo, FOO, etc., SHALL be considered equivalent. By convention, attributes will be written in all lowercase letters, with the underscore ("_") character used to separate distinct words within an attribute.

The following are examples of attribute-value pairs:

- color="red"
- vehicle_length="6"
- unit="meter"
- nickname="Zippy"

This document uses the following notation to illustrate WFNs:

`wfn:[a1=v1, a2=v2, …, an=vn]`

That is, WFNs will be notated as lists of attribute-value pairs enclosed in square brackets, prefixed with the string "wfn:". This notation is used solely for the purposes of explaining and illustrating the concepts and procedures specified herein. There is no requirement that implementations represent WFNs explicitly or use this notation in any way.

5.2 WFN Attributes

WFNs MUST satisfy these criteria:

1. Only the following attributes SHALL be permitted in a WFN attribute-value pair:
   a. part
   b. vendor
   c. product
   d. version
   e. update
   f. edition
   g. language
   h. sw_edition
   i. target_sw
   j. target_hw
   k. other
2. Each permitted attribute MAY be used at most once in a WFN. If an attribute is not used in a WFN, it is said to be *unspecified*, and its value SHALL default to the logical value ANY (cf. 5.3.1).
3. Attribute values of WFNs MUST satisfy the requirements specified in Section 5.3.

The attributes *sw_edition*, *target_sw*, *target_hw*, and *other* are newly introduced in this specification and are referred to collectively as the *extended attributes*.

5.3 WFN Attribute Values

Attributes of WFNs SHALL be assigned one of the following values:

1. A logical value specified in Section 5.3.1
2. A character string satisfying both (a) the requirements on string values specified in Section 5.3.2, and (b) the per-attribute value restrictions specified in Section 5.5.3.

5.3.1 Logical values

An attribute of a WFN MAY be assigned one of these logical values:

1. ANY (i.e., "any value"). The logical value ANY SHOULD be assigned to an attribute when there are no restrictions on acceptable values for that attribute.
2. NA (i.e., "not applicable/not used"). The logical value NA SHOULD be assigned when there is no legal or meaningful value for that attribute, or when that attribute is not used as part of the description. This includes the situation in which an attribute has an obtainable value that is null.

In WFNs, these logical values will be written in all uppercase characters, without surrounding quotation marks, on the right side of the equal sign, as in these examples:

- `wfn:[…,update=ANY,…]`
- `wfn:[…,update=NA,…]`

5.3.2 Restrictions on attribute-value strings

Value strings assigned to attributes of WFNs SHALL be non-empty contiguous strings of bytes encoded using printable Unicode Transformation Format-8 (UTF-8) characters with hexadecimal values between x00 and x7F [RFC3629].

In WFNs, attribute-value strings will be enclosed in double quotes as in the examples below. The quotation marks are, of course, not considered part of the string values themselves.

- `wfn:[…,update="sr1",…]`
- `wfn:[…,target_hw="x64",update="sp2"]`

Value strings in WFNs SHALL satisfy all of the following requirements:

1. Uppercase letters, lowercase letters or digit characters MAY be used (ASCII x41-x5a, x30-x39 and x61-x7a).
2. The *underscore* (x5f) MAY be used, and it SHOULD be used in place of whitespace characters (which SHALL NOT be used).
3. The backslash (x5c) MAY be used, and is designated the *escape character*. It SHOULD be used in a value string when required to modify the interpretation of the character that immediately follows (see below). In these circumstances, the character following the backslash is said to be *quoted*.
4. The *asterisk* (x2a) and the *question mark* (x3f) MAY be used, and are designated *special characters*. These two characters MAY be assigned special interpretations by other CPE specifications. To block special interpretation of these characters, precede them with the escape character, otherwise leave them unquoted in the value string. A special character MAY be included at the beginning of an otherwise non-empty value string, and/or at the end of the string. A special character MUST NOT be embedded within a value string. A single asterisk MUST NOT be used by itself as an attribute value. The asterisk MUST NOT be used more than once in sequence. A single question mark MAY be used by itself as an attribute value. The question mark MAY be used more than once in sequence.
5. All other *printable non-alphanumeric characters* (i.e., all punctuation marks, brackets, delimiters and other special purpose symbols, except for the special characters defined above) MUST be quoted when embedded in value strings.
6. A quoted hyphen MUST NOT be used by itself as a value string.

These requirements are summarized by the Augmented Backus-Naur Form (ABNF) grammar [RFC2234] for *avstring* shown in Figure 5-1.

```
avstring      = ( body / ( spec_chrs *body2 )) *1spec_chrs
spec_chrs     = 1*spec1 / spec2
spec1         = "?"
spec2         = "*"
body          = ( body1 *body2 ) / 2*body2
body1         = unreserved / quoted1
body2         = unreserved / quoted2
unreserved    = ALPHA / DIGIT / "_"
quoted1       = escape (escape / special / punc-no-dash)
quoted2       = escape (escape / special / punc-w-dash)
escape        = "\"
special       = spec1 / spec2
dash          = "-"
punc-no-dash  = "!" / DQUOTE / "#"/ "$"/ "%" / "&" / "'" /
                "(" / ")" / "+" / "," / "." / "/" /
                ":" / ";" / "<" / "=" / ">" / "@" / "[" /
                "]" / "^" / "`" / "{" / "|" / "}" / "~"
punc-w-dash   = punc-no-dash / dash
DQUOTE        = %x22 ; double quote
ALPHA         = %x41-5a / %x61-7A   ; A-Z or a-z
DIGIT         = %x30-39    ; 0-9
```

Figure 5-1. ABNF Grammar for Attribute-Value Strings

The following are examples of allowable value strings in WFNs:

- `"foo\-bar"` (hyphen is quoted)
- `"Acrobat_Reader"`
- `"\"oh_my\!\""` (quotation marks and exclamation point are quoted)
- `"g\+\+"` (plus signs are quoted)
- `"9\.?"` (period is quoted, question mark is unquoted)
- `"sr*"` (asterisk is unquoted)
- `"big\$money"` (dollar sign is quoted)
- `"foo\:bar"` (colon is quoted)
- `"back\\slash_software"` (backslash is quoted)
- `"with_quoted\~tilde"` (tilde is quoted)
- `"*SOFT*"` (single unquoted asterisk at beginning and end)
- `"8\.??"` (two unquoted question marks at end)
- `"*8\.??"` (one unquoted asterisk at beginning, two unquoted question marks at end)

5.3.3 Per-attribute value restrictions

This section specifies value restrictions on each WFN attribute and provides recommendations for how suitable value strings should be chosen.

5.3.3.1 Part

The *part* attribute SHALL have one of these three string values:

- The value "a", when the WFN is for a class of applications.
- The value "o", when the WFN is for a class of operating systems.
- The value "h", when the WFN is for a class of hardware devices.

5.3.3.2 Vendor

Values for this attribute SHOULD describe or identify the person or organization that manufactured or created the product. Values for this attribute SHOULD be selected from an attribute-specific valid-values list, which MAY be defined by other specifications that utilize this specification. Any character string meeting the requirements for WFNs (cf. 5.3.2) MAY be specified as the value of the attribute.

5.3.3.3 Product

Values for this attribute SHOULD describe or identify the most common and recognizable title or name of the product. Values for this attribute SHOULD be selected from an attribute-specific valid-values list, which MAY be defined by other specifications that utilize this specification. Any character string meeting the requirements for WFNs (cf. 5.3.2) MAY be specified as the value of the attribute.

5.3.3.4 Version

Values for this attribute SHOULD be vendor-specific alphanumeric strings characterizing the particular release version of the product. Version information SHOULD be copied directly (with escaping of printable non-alphanumeric characters as required) from discoverable data and SHOULD NOT be truncated or otherwise modified. Any character string meeting the requirements for WFNs (cf. 5.3.2) MAY be specified as the value of the attribute.

5.3.3.5 Update

Values for this attribute SHOULD be vendor-specific alphanumeric strings characterizing the particular update, service pack, or point release of the product. Values for this attribute SHOULD be selected from an attribute-specific valid-values list, which MAY be defined by other specifications that utilize this specification. Any character string meeting the requirements for WFNs (cf. 5.3.2) MAY be specified as the value of the attribute.

5.3.3.6 Edition

The *edition* attribute is considered deprecated in this specification, and it SHOULD be assigned the logical value ANY except where required for backward compatibility with version 2.2 of the CPE specification. This attribute is referred to as the "legacy *edition*" attribute.

If this attribute is used, values for this attribute SHOULD capture edition-related terms applied by the vendor to the product. Values for this attribute SHOULD be selected from an attribute-specific valid-values list, which MAY be defined by other specifications that utilize this specification. Any character string meeting the requirements for WFNs (cf. 5.3.2) MAY be specified as the value of the attribute.

5.3.3.7 SW_Edition

Values for this attribute SHOULD characterize how the product is tailored to a particular market or class of end users. Values for this attribute SHOULD be selected from an attribute-specific valid-values list, which MAY be defined by other specifications that utilize this specification. Any character string meeting the requirements for WFNs (cf. 5.3.2) MAY be specified as the value of the attribute.

5.3.3.8 Target_SW

Values for this attribute SHOULD characterize the software computing environment within which the product operates. Values for this attribute SHOULD be selected from an attribute-specific valid-values list, which MAY be defined by other specifications that utilize this specification. Any character string meeting the requirements for WFNs (cf. 5.3.2) MAY be specified as the value of the attribute.

5.3.3.9 Target_HW

Values for this attribute SHOULD characterize the instruction set architecture (e.g., x86) on which the product being described or identified by the WFN operates. Bytecode-intermediate languages, such as Java bytecode for the Java Virtual Machine or Microsoft Common Intermediate Language for the Common Language Runtime virtual machine, SHALL be considered instruction set architectures. Values for this attribute SHOULD be selected from an attribute-specific valid-values list, which MAY be defined by other specifications that utilize this specification. Any character string meeting the requirements for WFNs (cf. 5.3.2) MAY be specified as the value of the attribute.

5.3.3.10 Language

Values for this attribute SHALL be valid language tags as defined by [RFC5646], and SHOULD be used to define the language supported in the user interface of the product being described. Although any valid language tag MAY be used, only tags containing language and region codes SHOULD be used.

5.3.3.11 Other

Values for this attribute SHOULD capture any other general descriptive or identifying information which is vendor- or product-specific and which does not logically fit in any other attribute value. Values SHOULD NOT be used for storing instance-specific data (e.g., globally-unique identifiers or Internet Protocol addresses). Values for this attribute SHOULD be selected from a valid-values list that is refined over time; this list MAY be defined by other specifications that utilize this specification. Any character string meeting the requirements for WFNs (cf. 5.3.2) MAY be specified as the value of the attribute.

5.4 Operations on WFNs

Three functions are defined over WFNs: *new*, *get*, and *set*. These functions are used in defining binding and unbinding procedures in Section 6.

5.4.1 Function new()

The `new()` function takes no arguments. It returns an *empty WFN* (a WFN containing no attribute-value pairs).

Example:

```
new() → wfn:[]
```

5.4.2 Function get(w,a)

The `get(w,a)` function takes two arguments, a WFN *w* and an attribute *a*, and returns the value of *a*. If the attribute *a* is unspecified in *w*, `get(w,a)` returns the default value ANY.

Examples:

- `get(wfn:[vendor="microsoft",product="internet_explorer"],vendor) → "microsoft"`
- `get(wfn:[vendor="microsoft",product="internet_explorer"],version) → ANY`

5.4.3 Function set(w,a,v)

The `set(w,a,v)` function takes three arguments, a WFN *w*, an attribute *a*, and a value *v*. If the attribute *a* is unspecified in *w*, `set(w,a,v)` adds the attribute-value pair *a*=*v* to *w*. If the attribute *a* is specified in *w*, `set(w,a,v)` replaces its value with *v*. If *v* is *nil*, `set(w,a,v)` *deletes a* from *w* if *a* is specified in *w*, otherwise has no effect. The function always returns the new value of *w*.

Examples:

- `set(wfn:[],vendor,"microsoft") → wfn:[vendor="microsoft"]`
- `set(wfn:[vendor="microsoft"],vendor,"adobe") → wfn:[vendor="adobe"]`
- `set(wfn:[vendor="microsoft"],update,ANY) → wfn:[vendor="microsoft",update=ANY]`
- `set(wfn:[vendor="microsoft"],vendor,nil) → wfn:[]`

5.5 WFN Examples

This section illustrates a variety of WFNs. The examples below are intended only to illustrate names that are well formed according to the rules defined above. These examples do not necessarily illustrate "correct" or "valid" assignments of values to attributes.

- Microsoft Internet Explorer 8.0.6001 Beta (no edition):
  `wfn:[part="a",vendor="microsoft",product="internet_explorer", version="8\.0\.6001",update="beta",edition=NA]`

- Microsoft Internet Explorer 8.* SP? (no edition, any language):
  `wfn:[part="a",vendor="microsoft",product="internet_explorer", version="8\.*",update="sp?",edition=NA,language=ANY]`

- Identifier for HP Insight Diagnostics 7.4.0.1570 Online Edition for Windows 2003 x64:
  `wfn:[part="a",vendor="hp",product="insight_diagnostics", version="7\.4\.0\.1570",sw_edition="online", target_sw="windows_2003",target_hw="x64"]`

- Identifier for HP OpenView Network Manager 7.51 (no update) for Linux:
  ```
  wfn:[part="a",vendor="hp",product="openview_network_manager",
  version="7\.51",update=NA,target_sw="linux"]
  ```

- Foo\Bar Systems Big$Money 2010 Special Edition for iPod Touch:
  ```
  wfn:[part="a",vendor="foo\\bar",product="big\$money_2010",
  sw_edition="special",target_sw="ipod_touch"]
  ```

6. Name Binding and Unbinding

This section defines the procedures for *binding* (cf. 2.1) and *unbinding* (cf. 2.1) WFNs to and from machine-readable representations. Section 6.1 addresses URI binding/unbinding, and Section 6.2 covers formatted string binding/unbinding.

This section defines the formats for acceptable bound CPE names. The XML schema implementation of these format requirements, which is the authoritative XML binding definition, can be found at http://scap.nist.gov/schema/cpe/2.3/cpe-naming_2.3.xsd.

6.1 URI Binding

The URI Binding is included in this specification for backward compatibility with prior CPE versions. Section 5.1 of [CPE22] specifies that a CPE name is a percent-encoded URI [RFC3986] with each name having the URI scheme name "cpe:". The procedure defined here for creating a URI binding ensures that when a WFN is bound to a URI, it will satisfy the requirements of [CPE22] for CPE names. Section 6.1.1 defines the syntax of a valid URI binding. Section 6.1.2 specifies the procedure for binding a WFN to a URI. Section 6.1.3 specifies the procedure for unbinding a URI into a WFN.

6.1.1 URI Binding Syntax

The syntax of legal CPE v2.2 URIs is specified in Appendix A of [CPE22]. For ease of reference it is shown in Figure 6-1, using ABNF notation [RFC2234].

```
cpe-name          = "cpe:/" component-list

component-list    = part ":" vendor ":" product ":" version ":" update ":"
                    edition ":" lang
component-list    /= part ":" vendor ":" product ":" version ":" update ":"
                     edition
component-list    /= part ":" vendor ":" product ":" version ":" update
component-list    /= part ":" vendor ":" product ":" version
component-list    /= part ":" vendor ":" product
component-list    /= part ":" vendor
component-list    /= part
component-list    /= empty
part              = "h" / "o" / "a" / empty
vendor            = string
product           = string
version           = string
update            = string
edition           = string
lang              = LANGTAG / empty
string            = *( unreserved / pct-encoded )
empty             = ""
unreserved        = ALPHA / DIGIT / "-" / "." / "_" / "~"
pct-encoded       = "%" HEXDIG HEXDIG
ALPHA             = %x41-5a / %x61-7a   ; A-Z or a-z
DIGIT             = %x30-39  ; 0-9
HEXDIG            = DIGIT / "a" / "b" / "c" / "d" / "e" / "f"
LANGTAG           = cf. [RFC5646]
```

Figure 6-1. ABNF for v2.2-Compatible URI Binding

This syntax specification is less restrictive than suggested by other text in [CPE22] as well as current CPE operational practice. Specifically, it allows any character in the range %01-ff to be percent-encoded, but Section 5.4 of [CPE22] suggests that only certain printable non-alphanumeric characters should be percent-encoded. In practice, CPE names appearing in the v2.2 Official CPE Dictionary apply percent-encoding only to a limited set of printable non-alphanumeric characters.

To ensure full backward compatibility with [CPE22], any implementation that consumes URIs and claims to be conformant with this Naming specification MUST accept as valid input any URI that obeys the full syntax shown in Figure 6-1.

```
cpe-name          = "cpe:/" component-list

component-list    = part ":" vendor ":" product ":" version ":" update ":"
                    edition ":" lang
component-list    /= part ":" vendor ":" product ":" version ":" update ":"
                     edition
component-list    /= part ":" vendor ":" product ":" version ":" update
component-list    /= part ":" vendor ":" product ":" version
component-list    /= part ":" vendor ":" product
component-list    /= part ":" vendor
component-list    /= part
component-list    /= empty
part              = "h" / "o" / "a" / empty
vendor            = string
product           = string
version           = string
update            = string
edition           = string / packed
lang              = LANGTAG / empty
string            = str_wo_special / str_w_special
str_wo_special    = *( unreserved / pct-encoded )
str_w_special     = *1spec_chrs 1*( unreserved / pct-encoded ) *1spec_chrs
spec_chrs         = 1*spec1 / spec2
packed            = "~" string "~" string "~" string "~" string "~" string
empty             = ""
unreserved        = ALPHA / DIGIT / "-" / "." / "_"
special           = spec1 / spec2
spec1             = "%01"
spec2             = "%02"
pct-encoded       = "%21" / "%22" / "%23" / "%24" / "%25" / "%26" / "%27" /
                  = "%28" / "%29" / "%2a" / "%2b" / "%2c" / "%2f" / "%3a" /
                  = "%3b" / "%3c" / "%3d" / "%3e" / "%3f" / "%40" / "%5b" /
                  = "%5c" / "%5d" / "%5e" / "%60" / "%7b" / "%7c" / "%7d" /
                  = "%7e"
ALPHA             = %x41-5a / %x61-7a  ; A-Z or a-z
DIGIT             = %x30-39  ; 0-9
LANGTAG           = language ["-" region] ; cf. [RFC5646]
language          = 2*3ALPHA              ; shortest ISO 639 code
region            = 2ALPHA / 3DIGIT       ; ISO 3166-1 code or UN M.49 code
```

Figure 6-2. ABNF for Preferred URI Binding

Although the freer syntax is allowed as valid input, CPE URIs SHOULD obey the more restricted syntax shown in Figure 6-2. The syntax shown in Figure 6-2 has the following features:

- It introduces support for "packing" multiple attribute values into the edition component.
- It introduces support for the use of special characters.
- It strictly enumerates the allowed percent-encoded forms.

The `bind_to_URI` procedure (cf. 6.1.2) is designed to produce URIs that obey the syntax shown in Figure 6-2. The `unbind_URI` procedure (cf. 6.1.3) is designed to accept as valid input URIs that obey the syntax shown in either Figure 6-1 or Figure 6-2.

6.1.2 Binding a WFN to a URI

The procedure to bind a given WFN to a URI is specified in pseudocode below. The top-level binding function, `bind_to_URI`, is called with the WFN to be bound as its only argument. The pseudocode references the defined operations on WFNs (cf. 5.4) as well as a number of helper functions also defined in pseudocode. Section 6.1.2.1 provides some important notes on the binding procedure. Section 6.1.2.2 summarizes the algorithm in prose. Section 6.1.2.3 provides the pseudocode for the algorithm. Section 6.1.2.4 provides examples of binding WFNs to URIs. The algorithm defined here assumes that the input WFN complies with the requirements defined in Section 5.2. The behavior of `bind_to_URI` is undefined if its input is not well formed.

It is OPTIONAL for implementations of URI binding to follow the pseudocode, processes, and other non-normative information provided in this section. However, the result of an implementation performing a URI binding MUST produce the same result as the suggested pseudocode, processes, etc. would produce.

6.1.2.1 Notes on URI binding procedure

The procedure for binding WFNs to URIs MUST address the following issues:

1. Handling logical values
2. Handling non-alphanumeric and special characters
3. "Packing" extended attributes.

6.1.2.1.1 Handling logical values

The logical values ANY and NA are defined as possible attribute values in WFNs. The logical value ANY SHALL bind to what [CPE22] calls a "blank" (i.e., an empty component, indicated by two sequential colons) in the URI. The logical value NA SHALL bind to a single hyphen.

6.1.2.1.2 Handling non-alphanumeric and special characters

In WFN attribute-value strings, non-alphanumeric characters generally MUST be quoted, with the exception that the special characters "*" and "?" MAY appear without quoting. When forming the URI binding, certain transformations MUST be applied. In most cases, quoted non-alphanumeric characters are bound to a percent-encoded form; certain quoted non-alphanumeric characters, however, are passed through without percent-encoding. When special characters appear without quoting, they are mapped to special percent-encoded forms. This behavior is summarized as follows.

Following Table 6-1, the binding procedure SHALL replace quoted non-alphanumeric substrings shown under the columns labeled IN with the substitute character or percent-encoded string shown to its right under the columns labeled OUT. So, for example, if the substring `"\(` (exclusive of double quotes) is

encountered within an attribute-value string, it would be bound to the substring "%28". As a second example, if the substring "\-" is encountered within an attribute-value string, it would be bound to the single character "-".

IN	OUT	IN	OUT	IN	OUT	IN	OUT	IN	OUT	IN	OUT	IN	OUT
\!	%21	\"	%22	\#	%23	\$	%24	\%	%25	\&	%26	\'	%27
\(	%28	\)	%29	*	%2a	\+	%2b	\,	%2c	\-	-	\.	.
\/	%2f	\:	%3a	\;	%3b	\<	%3c	\=	%3d	\>	%3e	\?	%3f
\@	%40	\[	%5b	\\	%5c	\]	%5d	\^	%5e	\`	%60	\{	%7b
|	%7c	\}	%7d	\~	%7e								

Table 6-1. Bindings for Quoted Non-Alphanumeric Characters

When an unquoted special character is encountered in an attribute-value string, it SHALL be bound to the special percent-encoded form shown in Table 6-2.

IN	OUT
?	%01
*	%02

Table 6-2. Bindings for Unquoted Special Characters

6.1.2.1.3 Packing extended attributes

The version 2.3 specification introduces four new attributes—the extended attributes—that have no assigned position in the URI binding. When these attributes have values other than ANY in the WFN, they are "packed" in a special format, and in a specified order, into the edition component of the URI. This special format uses the tilde character "~" (without percent-encoding) as a sub-delimiter.

The capability to bind WFNs to URIs is provided primarily for use by CPE dictionary creators and maintainers, to allow them to create new CPE names that take full advantage of all features introduced in this specification, while still having a backward-compatible path for creating approximate names that conform to [CPE22]. This capability should be used with care, as CPE v2.2-conformant tools may be unable to properly match names that differ in terms of packed attribute values.

6.1.2.2 Summary of algorithm

The URI binding procedure is summarized as follows:

1. Initialize the output URI binding to the string "cpe:/".
2. Begin loop: Iterate over the seven attributes corresponding to the seven components in a v2.2 CPE URI [CPE22:5.2]. Get the value of each attribute and perform steps 3 through 7.
3. Special handling of edition: When binding to a 2.2 URI, the edition component (the sixth element of the URI) is used as the location to "pack" five attribute values in the WFN: (legacy) edition, sw_edition, target_sw, target_hw, and other. The "packing" process involves concatenating the five values together, prefixed and separated by the tilde. The leading tilde serves as a flag indicating that the contents of the edition field are a packed representation of five separate values, and the internal tildes are used to aid parsing the values out. In the special case in which the four

extended attributes are not specified, or all are ANY, only the edition attribute is used and no packing is performed.

4. Bind attribute values:
    a. For all attributes *other than* (legacy) "edition", inspect the value and convert logical values appropriately. If the attribute is unspecified, or its logical value is ANY, bind it to blank ("") in the URI. If the logical value is NA, bind it to the hyphen ("-").
    b. Scan the attribute value for any quoted non-alphanumeric characters and bind them as specified in Table 6-1.
    c. Scan the attribute value for any unquoted special characters and bind them as specified in Table 6-2.
5. Append the attribute-value string to the output URI, followed by a trailing colon.
6. END LOOP.
7. Return the output URI, trimming away all trailing colons for compactness.

6.1.2.3 Pseudocode for algorithm

```
function bind_to_URI(w)
  ;; Top-level function used to bind a WFN w to a URI.
  ;; Initialize the output with the CPE v2.2 URI prefix.
  uri := "cpe:/".
  foreach a in {part,vendor,product,version,update,edition,language}
    do
      if a = edition
        then
          ;; Call the pack() helper function to compute the proper
          ;; binding for the edition element.
          ed := bind_value_for_URI(get(w,edition)).
          sw_ed := bind_value_for_URI(get(w,sw_edition)).
          t_sw := bind_value_for_URI(get(w,target_sw)).
          t_hw := bind_value_for_URI(get(w,target_hw)).
          oth := bind_value_for_URI(get(w,other)).
          v := pack(ed,sw_ed,t_sw,t_hw,oth).
        else
          ;; Get the value for a in w, then bind to a string
          ;; for inclusion in the URI.
          v := bind_value_for_URI(get(w,a)).
      endif.
    ;; Append v to the URI then add a colon.
    uri := strcat(uri, v, ":").
  end.
  ;; Return the URI string, with trailing colons trimmed.
  return trim(uri).
end.

function bind_value_for_URI(s)
  ;; Takes a string s and converts it to the proper string for
  ;; inclusion in a CPE v2.2-conformant URI.  The logical value ANY
  ;; binds to the blank in the 2.2-conformant URI.
  if s = ANY then return("").
  ;; The value NA binds to a single hyphen.
  if s = NA then return("-").
```

```
  ;; If we get here, we're dealing with a string value.
  return transform_for_uri(s).
end.

function transform_for_uri(s)
  ;; Scans an input string s and applies the following
  ;; transformations:
  ;; - Pass alphanumeric characters thru untouched
  ;; - Percent-encode quoted non-alphanumerics as needed
  ;; - Unquoted special characters are mapped to their special forms.
  result := "".
  idx := 0.
  while (idx < strlen(s)) do
    thischar := substr(s,idx,idx).  ; get the idx'th character of s.
    ;; alphanumerics (incl. underscore) pass untouched.
    if (is_alphanum(thischar))
      then
        result := strcat(result,thischar).
        idx := idx + 1.
        continue.
    endif.
    if (thischar = "\") ; escape character
      then
        idx := idx + 1.
        nxtchar := substr(s,idx,idx).
        result := strcat(result,pct_encode(nxtchar)).
        idx := idx + 1.
        continue.
    endif.
    ;; Bind the unquoted '?' special character to "%01".
    if (thischar = "?") then result := strcat(result,"%01").
    ;; Bind the unquoted '*' special character to "%02".
    if (thischar = "*") then result := strcat(result,"%02").
    idx := idx + 1.
  end.
  return result.
end.

function pct_encode(c)
  ;; Return the appropriate percent-encoding of character c.
  ;; Certain characters are returned without encoding.
  case c:
    '!': return "%21".
    '"': return "%22".
    '#': return "%23".
    '$': return "%24".
    '%': return "%25".
    '&': return "%26".
    ''': return "%27".
    '(': return "%28".
    ')': return "%29".
    '*': return "%2a".
```

```
    '+': return "%2b".
    ',': return "%2c".
    '-': return c. ; bound without encoding
    '.': return c. ; bound without encoding
    '/': return "%2f".
    ':': return "%3a".
    ';': return "%3b".
    '<': return "%3c".
    '=': return "%3d".
    '>': return "%3e".
    '?': return "%3f".
    '@': return "%40".
    '[': return "%5b".
    '\': return "%5c".
    ']': return "%5d".
    '^': return "%5e".
    '`': return "%60".
    '{': return "%7b".
    '|': return "%7c".
    '}': return "%7d".
    '~': return "%7e".
  end case.
end.

function pack(ed,sw_ed,t_sw,t_hw,oth)
  ;; "Pack" the values of the five arguments into the single edition
  ;; component. If all the values are blank, just return a blank.
  if (sw_ed = "" and t_sw = "" and t_hw = "" and oth = "")
    then
      ;; All the extended attributes are blank, so don't do
      ;; any packing, just return ed.
      return ed.
  end.
  ;; Otherwise, pack the five values into a single string
  ;; prefixed and internally delimited with the tilde.
  return strcat('~',ed,'~',sw_ed,'~',t_sw,'~',t_hw,'~',oth).
end.

function trim(s)
  ;; Remove trailing colons from the URI back to the first non-colon.
  s1 := reverse(s).
  idx := 0.
  for i := 0 to strlen(s1) do
    if substr(s1,i,i) = ":"
      then idx := idx + 1.
      else break.
  end.
  ;; Return the substring after all trailing colons,
  ;; reversed back to its original character order.
  return reverse(substr(s1,idx,strlen(s1)-1)).
end.
```

```
function strcat(s1,s2,...sn)
  ;; Returns a copy of the string s1 with the strings s2 to sn
  ;; appended in the order given.
  ;; Cf. the GNU C definition of strcat. This function shown
  ;; here differs only in that it can take a variable number
  ;; of arguments. This is really just shorthand for,
  ;; strcat(s1, strcat(s2, strcat(s3, … ))).
end.

function strlen(s)
  ;; Defined as in GNU C, returns the length of string s.
  ;; Returns zero if the string is empty.
end.

function substr(s,b,e)
  ;; Returns a substring of s, beginning at the b'th character,
  ;; with 0 being the first character, and ending at the e'th
  ;; character. b must be <= e. Returns nil if b >= strlen(s).
end.

function reverse(s)
  ;; Returns a reverse copy of string s, i.e., the last character
  ;; becomes the first character, the second-to-last becomes the
  ;; second character, etc.
end.

function is_alphanum(c)
  ;; Returns TRUE iff c is an uppercase letter, a lowercase letter,
  ;; a digit, or the underscore, otherwise FALSE.
end.
```

6.1.2.4 Examples of binding a WFN to a URI

This section presents several examples of binding WFNs to URIs.

6.1.2.4.1 Example 1

Suppose one had created the WFN below to describe this product: Microsoft Internet Explorer 8.0.6001 Beta (any edition):

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update="beta",edition=ANY]
```

This WFN binds to the following URI:

```
cpe:/a:microsoft:internet_explorer:8.0.6001:beta
```

Note how the trailing colons are removed, such that the "edition=ANY" effectively disappears.

6.1.2.4.2 Example 2

Suppose one had created the WFN below to describe this product: Microsoft Internet Explorer 8.* SP?:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.*",update="sp?"]
```

This WFN binds to the following URI:

```
cpe:/a:microsoft:internet_explorer:8.%02:sp%01
```

Note how the unquoted special characters in the WFN get percent-encoded in the URI. The otherwise prohibited percent-encoded forms "%01" and "%02" are used in the URI to represent the unquoted "?" and "*", respectively.

6.1.2.4.3 Example 3

Suppose one had created the WFN below to describe this product: HP Insight Diagnostics 7.4.0.1570 Online Edition for Windows 2003 x64:

```
wfn:[part="a",vendor="hp",product="insight_diagnostics",
version="7\.4\.0\.1570",update=NA,
sw_edition="online",target_sw="win2003",target_hw="x64"]
```

This WFN binds to the following URI:

```
cpe:/a:hp:insight_diagnostics:7.4.0.1570:-:~~online~win2003~x64~
```

Note how the legacy edition attribute as well as the four extended attributes are packed into the edition component of the URI.

6.1.2.4.4 Example 4

Suppose one had created the WFN below to describe this product: HP OpenView Network Manager 7.51 (any update) for Linux:

```
wfn:[part="a",vendor="hp",product="openview_network_manager",
version="7\.51",target_sw="linux"]
```

This WFN binds to the following URI:

```
cpe:/a:hp:openview_network_manager:7.51::~~~linux~~
```

Note how the unspecified update attribute binds to a blank in the URI, and how packing occurs in the edition component when only the target_sw attribute is specified.

6.1.2.4.5 Example 5

Suppose one had created the WFN below to describe this product: Foo\Bar Big$Money Manager 2010 Special Edition for iPod Touch 80GB:

```
wfn:[part="a",vendor="foo\\bar",product="big\$money_manager_2010",
sw_edition="special",target_sw="ipod_touch",target_hw="80gb"]
```

This WFN binds to the following URI:

```
cpe:/a:foo%5cbar:big%24money_manager_2010:::~~special~ipod_touch~80gb~
```

Note how the \\ becomes a percent-encoded backslash. Also note how the dollar sign is percent-encoded, and how the extended attributes are packed.

6.1.3 Unbinding a URI to a WFN

Given a CPE v2.2-conformant URI, the procedure to unbind it to a WFN is specified in pseudocode below. The top-level unbinding function, `unbind_URI`, is called with the URI to be unbound as its only argument. The pseudocode references the defined operations on WFNs (cf. 5.4) as well as a number of helper functions also defined in pseudocode. Section 6.1.3.1 summarizes the algorithm in prose. Section 6.1.3.2 provides the pseudocode for the algorithm. Section 6.1.3.3 provides examples of unbinding URIs to WFNs. Note that the pseudocode below reuses a number of helper functions defined above in Section 6.1.2.3. The algorithm expects that the syntax of the input URI conforms to the ABNF in Figure 6-1 or Figure 6-2. (This is guaranteed if the URI is the result of binding a WFN.) The algorithm performs some limited error checking and handling: (1) if an illegal percent-encoded form is encountered, an error is raised; (2) if unquoted special characters are used in a way that violates the legal syntax, an error is raised. A fully robust implementation of the unbinding procedure SHOULD implement additional syntax checks, such as making sure that non-alphanumeric characters do not appear without percent-encoding.

It is OPTIONAL for implementations of URI unbinding to follow the pseudocode, processes, and other non-normative information provided in this section. However, the result of an implementation performing a URI unbinding MUST produce the same result as the suggested pseudocode, processes, etc. would produce.

6.1.3.1 Summary of algorithm

The procedure for unbinding a URI is straightforward:

1. Loop over the seven attributes corresponding to the seven CPE v2.2 components, performing steps 2 through 7.
2. For URI components 1-5 and 7, decode the string and set the corresponding WFN attribute value. Decoding entails: converting sole "" to ANY, sole "-" to NA, adding quoting to embedded periods and hyphens, and decoding percent-encoded forms.
3. The edition component is "unpacked" if a leading tilde indicates it contains a packed collection of five attribute values.

6.1.3.2 Pseudocode for algorithm

```
function unbind_URI(uri)
  ;; Top-level function used to unbind a URI uri to a WFN.
  ;; Initialize the empty WFN.
  result := new().
  for i := 1 to 7
    do
      v := get_comp_uri(uri,i). ; get the i'th component of uri
      ;; unbind the parsed string.
      case i:
        1: result := set(result,part,decode(v)).
        2: result := set(result,vendor,decode(v)).
        3: result := set(result,product,decode(v)).
        4: result := set(result,version,decode(v)).
        5: result := set(result,update,decode(v)).
```

```
        6: ;; Special handling for edition component.
           ;; Unpack edition if needed.
           if (v = "" or v = "-" or substr(v,0,0) != "~")
             then
               ;; Just a logical value or a non-packed value.
               ;; So unbind to legacy edition, leaving other
               ;; extended attributes unspecified.
               result := set(result,edition,decode(v)).
             else
               ;; We have five values packed together here
               result := unpack(v,result).
           end.
        7: result := set(result,language,decode(v)).
      end.
  end.
  return result.
end.

function get_comp_uri(uri,i)
  ;; Return the i'th CPE component of the URI. If i=0,
  ;; return the URI scheme. For example, given URI:
  ;; cpe:/a:foo::bar:-
  ;; get_comp_uri(uri,0) = "cpe:"
  ;; get_comp_uri(uri,1) = "a"
  ;; get_comp_uri(uri,2) = "foo"
  ;; get_comp_uri(uri,3) = ""
  ;; get_comp_uri(uri,4) = "bar"
  ;; get_comp_uri(uri,5) = "-"
  ;; get_comp_uri(uri,6) = ""
  ;; etc.
end.

function decode(s)
  ;; This function scans the string s and returns a copy
  ;; with all percent-encoded characters decoded. This
  ;; function is the inverse of pct_encode(s) defined in
  ;; Section 6.1.2.3. Only legal percent-encoded forms are
  ;; decoded.  Others raise an error.
  ;; Decode a blank to logical ANY, and hyphen to logical NA.
  if (s = '') then return ANY.
  if (s = '-') then return NA.
  ;; Start the scanning loop.
  ;; Normalize: convert all uppercase letters to lowercase first.
  s := to_lowercase(s).
  result := "".
  idx := 0.
  embedded := false.
  while (idx < strlen(s))
    do
      c := substr(s,idx,idx).  ; get the idx'th character of s.
      ;; deal with dot, hyphen and tilde: decode with quoting
      if ((c = '.') or (c = '-') or (c = '~')) then
```

```
  result := strcat(result,"\\",c).
  idx := idx + 1.
  embedded := true.  ; a non-%01 encountered.
  continue.
endif.
if (c != '%') then
  result := strcat(result,c).
  idx := idx + 1.
  embedded := true.  ; a non-%01 encountered.
  continue.
endif.
;; we get here if we have a substring starting w/ '%'.
form := substr(s,idx,idx+2).   ; get the three-char sequence
case form:
  ;; pseudocode below uses \ within quotes as escape char
  "%01": if ;; %01 legal at beginning or end
            (((idx = 0) or (idx = (strlen(s)-3))) or
             ;; embedded is false, so must be preceded by %01
             (!embedded and (substr(s,idx-3,idx-1) = "%01")) or
             ;; embedded is true, so must be followed by %01
             (embedded and (strlen(s) >= idx+6) and
              (substr(s,idx+3,idx+5) = "%01")))
           then
             ;; A percent-encoded question mark is found
             ;; at the beginning or the end of the string,
             ;; or embedded in sequence as required.
             ;; Decode to unquoted form.
             result := strcat(result,"?").
             idx := idx + 3.
             continue.
           else
             error.  ;; raise an exception
         endif.
  "%02": if ((idx = 0) or (idx = (strlen(s)-3)))
           then
             ;; Percent-encoded asterisk is at the beginning
             ;; or the end of the string, as required.
             ;; Decode to unquoted form.
             result := strcat(result,"*").
           else
             error.  ;; raise an exception
         endif.
  "%21": result := strcat(result,"\\!").
  "%22": result := strcat(result,"\\\"").
  "%23": result := strcat(result,"\\#").
  "%24": result := strcat(result,"\\$").
  "%25": result := strcat(result,"\\%").
  "%26": result := strcat(result,"\\&").
  "%27": result := strcat(result,"\\'").
  "%28": result := strcat(result,"\\(").
  "%29": result := strcat(result,"\\)").
  "%2a": result := strcat(result,"\\*").
```

```
        "%2b": result := strcat(result,"\\+").
        "%2c": result := strcat(result,"\\,").
        "%2f": result := strcat(result,"\\/").
        "%3a": result := strcat(result,"\\:").
        "%3b": result := strcat(result,"\\;").
        "%3c": result := strcat(result,"\\<").
        "%3d": result := strcat(result,"\\=").
        "%3e": result := strcat(result,"\\>").
        "%3f": result := strcat(result,"\\?").
        "%40": result := strcat(result,"\\@").
        "%5b": result := strcat(result,"\\[").
        "%5c": result := strcat(result,"\\\\").
        "%5d": result := strcat(result,"\\]").
        "%5e": result := strcat(result,"\\^").
        "%60": result := strcat(result,"\\`").
        "%7b": result := strcat(result,"\\{").
        "%7c": result := strcat(result,"\\|").
        "%7d": result := strcat(result,"\\}").
        "%7e": result := strcat(result,"\\~").
       else: error.  ; any other form is an error
      end case.
    idx := idx + 3.
    embedded := true. ; a non-%01 encountered.
  end.
  return result.
end.

function unpack(s,wfn).
  ;; Argument s is a packed edition string, wfn is a WFN.
  ;; Unpack its elements and set the attributes in wfn accordingly.
  ;; Parse out the five elements.
  start := 1.
  end := strchr(s,'~',start).
  if (start = end)
    then ed := "".
    else ed := substr(s,start,end-1).
  end.
  start := end+1.
  end := strchr(s,'~',start).
  if (start = end)
    then sw_ed := "".
    else sw_ed := substr(s,start,end-1).
  end.
  start := end+1.
  end := strchr(s,'~',start).
  if (start = end)
    then t_sw := "".
    else t_sw := substr(s,start,end-1).
  end.
  start := end+1.
  end := strchr(s,'~',start).
  if (start = end)
```

```
    then t_hw := "".
    else t_hw := substr(s,start,end-1).
  end.
  start := end+1.
  if (start >= strlen(s))
    then oth := "".
    else oth := substr(s,start,strlen(s)-1).
  end.
  wfn := set(wfn,edition,decode(ed)).
  wfn := set(wfn,sw_edition,decode(sw_ed)).
  wfn := set(wfn,target_sw,decode(t_sw)).
  wfn := set(wfn,target_hw,decode(t_hw)).
  wfn := set(wfn,other,decode(oth)).
  return wfn.
end.

function strchr(str,chr,off)
  ;; Searches the string str for the character chr starting
  ;; at offset off into the string. Returns the offset of
  ;; the chr if found, otherwise nil.
  ;; Defined similar to the standard C function strchr.
  ;; But this version takes a third argument off, which
  ;; is an offset into the str to begin the search.
end.

function to_lowercase(s)
  ;; convert all alphabetic characters to lowercase.
end.
```

6.1.3.3 Examples of unbinding a URI to a WFN

This section provides a number of examples illustrating the results of unbinding a URI to a WFN.

6.1.3.3.1 Example 1

URI: `cpe:/a:microsoft:internet_explorer:8.0.6001:beta`

Unbinds to this WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update="beta",edition=ANY,
language=ANY]
```

Notice how the legacy edition and language attributes are unbound to the logical value ANY.

6.1.3.3.2 Example 2

URI: `cpe:/a:microsoft:internet_explorer:8.%2a:sp%3f`

Unbinds to this WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.\*",update="sp\?",edition=ANY,language=ANY]
```

Note how the two percent-encoded characters are unbound with added quoting. Although the percent-encoded characters are the same as the special characters, the added quoting blocks their interpretation in the WFN.

6.1.3.3.3 Example 3

URI: `cpe:/a:microsoft:internet_explorer:8.%02:sp%01`

Unbinds to this WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.*",update="sp?",edition=ANY,language=ANY]
```

Note how the two percent-encoded special characters are unbound without quoting.

6.1.3.3.4 Example 4

URI: `cpe:/a:hp:insight_diagnostics:7.4.0.1570::~~online~win2003~x64~`

Unbinds to this WFN:

```
wfn:[part="a",vendor="hp",product="insight_diagnostics",
version="7\.4\.0\.1570",update=ANY,edition=ANY,
sw_edition="online",target_sw="win2003",target_hw="x64",
other=ANY,language=ANY]
```

Note how the legacy edition attribute as well as the four extended attributes are unpacked from the edition component of the URI.

6.1.3.3.5 Example 5

URI: `cpe:/a:hp:openview_network_manager:7.51:-:~~~linux~~`

Unbinds to this WFN:

```
wfn:[part="a",vendor="hp",product="openview_network_manager",
version="7\.51",update=NA,edition=ANY,sw_edition=ANY,
target_sw="linux",target_HW=ANY,other=ANY,language=ANY]
```

Note how the lone hyphen in the update component is unbound to the logical value NA, and how all the other blanks embedded in the packed edition component unbind to ANY, with only the target_sw attribute actually specified.

6.1.3.3.6 Example 6

URI: `cpe:/a:foo%5cbar:big%24money_2010%07:::~~special~ipod_touch~80gb~`

An error is raised when this URI is unbound, because it contains an illegal percent-encoded form, "%07".

6.1.3.3.7 Example 7

URI: `cpe:/a:foo~bar:big%7emoney_2010`

Unbinds to this WFN:

```
wfn:[part="a",vendor="foo\~bar",product="big\~money_2010",
version=ANY,update=ANY,edition=ANY,language=ANY]
```

Note how both the tildes (unencoded as well as percent-encoded) are handled: both are quoted in the WFN. The original v2.2 URI syntax allows tildes to appear without encoding, but the preferred URI syntax is for tildes to be encoded like any other printable non-alphanumeric character.

6.1.3.3.8 Example 8

URI: `cpe:/a:foo:bar:12.%02.1234`

An error is raised when this URI is unbound, because it contains a special character ("%02") embedded within a value string.

6.2 Formatted String Binding

The formatted string binding is new to CPE version 2.3. In keeping with the spirit of the v2.2 specification, the formatted string binding looks similar to the URI binding; however, it is defined simply to be a "formatted string" rather than a URI in order to relax the requirements that typically apply to URIs as specified in [RFC3986].

The formatted string binding is a colon-delimited list of fields prefixed with the string `"cpe:2.3:"`. Use of a prefix distinct from the v2.2 URI binding enables tools to inspect a given input string and use a simple syntactic test to determine whether to process the input as a URI or as a formatted string. The formal syntax of the formatted string binding is presented in ABNF in Section 6.2.1.

Similar to the URI binding, the formatted string binds the attributes in a WFN in a fixed order, separated by the colon character:

```
cpe:2.3: part : vendor : product : version : update : edition :
       language : sw_edition : target_sw : target_hw : other
```

In a formatted string binding, the alphanumeric characters plus hyphen ("-"), period (".") and underscore ("_") appear unquoted. When used alone, the asterisk ("*") represents the logical value ANY, and the hyphen ("-") represents the logical value NA. All other non-alphanumeric characters, if used, MUST be quoted (preceded by the backslash). The special characters asterisk and question mark may appear without quoting, in which case they are open to special interpretation by other CPE specifications. Note that all eleven (11) attribute values MUST appear in the formatted string binding.

6.2.1 Syntax for Formatted String Binding

The syntax of the formatted string binding is shown in Figure 6-3.

```
formstring      = "cpe:2.3:" component-list

component-list  = part ":" vendor ":" product ":" version ":" update ":"
                  edition ":" lang ":" sw_edition ":" target_sw ":"
                  target_hw ":" other

part            = "h" / "o" / "a" / logical
vendor          = avstring
product         = avstring
version         = avstring
update          = avstring
edition         = avstring
lang            = LANGTAG / logical
sw_edition      = avstring
target_sw       = avstring
target_hw       = avstring
other           = avstring
avstring        = *1spec_chrs 1*( unreserved / quoted ) *1spec_chrs / logical
spec_chrs       = 1*spec1 / spec2
logical         = "*" / "-"
special         = spec1 / spec2
spec1           = "?"
spec2           = "*"
unreserved      = ALPHA / DIGIT / "-" / "." / "_"
quoted          = escape (escape / special / punc)
escape          = "\"
punc            = "!" / DQUOTE / "#" / "$" / "%" / "&" / "'" / "(" / ")" /
                  "+" / "," / "/" / ":" / ";" / "<" / "=" / ">" / "@" /
                  "[" / "]" / "^" / "`" / "{" / "|" / "}" / "~"
ALPHA           = %x41-5a / %x61-7A   ; a-z
DIGIT           = %x30-39  ; 0-9
DQUOTE          = %x22  ; double-quote
LANGTAG         = language ["-" region] ; cf. [RFC5646]
language        = 2*3ALPHA              ; shortest ISO 639 code
region          = 2ALPHA / 3DIGIT       ; ISO 3166-1 code or UN M.49 code
```

Figure 6-3. ABNF for Formatted String Binding

6.2.2 Binding a WFN to a formatted string

This section specifies the procedure that SHALL be followed when binding a WFN to a formatted string. Section 6.2.2.1 summarizes the algorithm in prose. Section 6.2.2.2 presents the pseudocode for the algorithm. Section 6.2.2.3 presents examples illustrating the results of binding various WFNs to formatted strings.

It is OPTIONAL for implementations of formatted string binding to follow the pseudocode, processes, and other non-normative information provided in this section. However, the result of an implementation performing a formatted string binding MUST produce the same result as the suggested pseudocode, processes, etc. would produce.

6.2.2.1 Summary of algorithm

The procedure iterates over the eleven allowed attributes in a fixed order. Corresponding attribute values are obtained from the input WFN and conversions of logical values are applied. A result string is formed by concatenating the attribute values separated by colons.

6.2.2.2 Pseudocode for algorithm

```
function bind_to_fs(w)
  ;; Top-level function used to bind WFN w to formatted string.
  ;; Initialize the output with the CPE v2.3 string prefix.
  fs := "cpe:2.3:".
  foreach a in {part,vendor,product,version,update,edition,language,
                sw_edition,target_sw,target_hw,other}
    do
      v := bind_value_for_fs(get(w,a)).
      fs := strcat(fs,v).
      ;; add a colon except at the very end
      if (a != other) then fs := strcat(fs,":").
  end.
  return fs.
end.

function bind_value_for_fs(v)
  ;; Convert the value v to its proper string representation for
  ;; insertion into the formatted string.
  case v:
    ANY: return("*").
    NA: return("-").
    else: return process_quoted_chars(v).
  end.
end.

function process_quoted_chars(s)
  ;; Inspect each character in string s. Certain nonalpha
  ;; characters pass thru without escaping into the result,
  ;; but most retain escaping.
  result := "".
  idx := 0.
  while (idx < strlen(s))
    do
      c := substr(s,idx,idx).  ; get the idx'th character of s.
      if c != "\"
        then
          ;; unquoted characters pass thru unharmed
          result := strcat(result,c).
        else
          ;; Escaped characters are examined
          nextchr := substr(s,idx+1,idx+1).
          case nextchr:
            ;; the period, hyphen and underscore pass unharmed.
```

```
            ".":
            "-":
            "_": result := strcat(result,nextchr).
                 idx = idx + 2.
            else:
              ;; all others retain escaping
              result := strcat(result,"\",nextchr).
              idx := idx + 2.
              continue.
          end.
        endif.
        idx := idx + 1.
  end.
  return result.
end.
```

6.2.2.3 Examples of binding a WFN to a formatted string

This section presents examples illustrating the results of binding various WFNs to formatted strings.

6.2.2.3.1 Example 1

Suppose one had created the WFN below to describe this product: Microsoft Internet Explorer 8.0.6001 Beta (any edition):

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update="beta",edition=ANY]
```

This WFN binds to the following formatted string:

```
cpe:2.3:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:*
```

Note how the unspecified attributes bind to "*" in the formatted string binding.

6.2.2.3.2 Example 2

Suppose one had created the WFN below to describe this product: Microsoft Internet Explorer 8.* SP? (any edition):

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.*",update="sp?",edition=ANY]
```

This WFN binds to the following formatted string:

```
cpe:2.3:a:microsoft:internet_explorer:8.*:sp?:*:*:*:*:*:*
```

Note how the unspecified attributes default to ANY and are thus bound to "*". Also note how the unquoted special characters in the WFN are carried over into the formatted string. Their special functionality in the WFN is preserved in the binding. If it was desired to block the special interpretation of the asterisk, it should be preceded by the escape character in the WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.\*",update="sp?"]
```

This WFN binds to the following formatted string:

```
cpe:2.3:a:microsoft:internet_explorer:8.\*:sp?:*:*:*:*:*:*
```

In this case, the escape character appears explicitly in the binding, blocking the interpretation of the asterisk. The unquoted question mark retains any special interpretation it may have in the binding.

6.2.2.3.3 Example 3

Suppose one had created the WFN below to describe this product: HP Insight 7.4.0.1570 Online Edition for Windows 2003 x64:

```
wfn:[part="a",vendor="hp",product="insight",
version="7\.4\.0\.1570",update=NA,
sw_edition="online",target_sw="win2003",target_hw="x64"]
```

This WFN binds to the following formatted string:

```
cpe:2.3:a:hp:insight:7.4.0.1570:-:*:*:online:win2003:x64:*
```

Notice how the NA binds to the lone hyphen, the unspecified edition, language and other all bind to the asterisk, and the extended attributes appear in their own fields.

6.2.2.3.4 Example 4

Suppose one had created the WFN below to describe this product: HP OpenView Network Manager 7.51 (any update) for Linux:

```
wfn:[part="a",vendor="hp",product="openview_network_manager",
version="7\.51",target_sw="linux"]
```

This WFN binds to the following formatted string:

```
cpe:2.3:a:hp:openview_network_manager:7.51:*:*:*:*:linux:*:*
```

Note how the unspecified attributes update, edition, language, sw_edition, target_hw, and other each bind to an asterisk in the formatted string.

6.2.2.3.5 Example 5

Suppose one had created the WFN below to describe this product: Foo\Bar Big$Money 2010 Special Edition for iPod Touch 80GB:

```
wfn:[part="a",vendor="foo\\bar",product="big\$money_2010",
sw_edition="special",target_sw="ipod_touch",target_hw="80gb"]
```

This WFN binds to the following formatted string:

```
cpe:2.3:a:foo\\bar:big\$money_2010:*:*:*:*:special:ipod_touch:80gb:*
```

Note how the \\ and \$ carry over into the binding, and how all the other unspecified attributes bind to the asterisk.

6.2.3 Unbinding a formatted string to a WFN

Given a formatted string binding, the procedure that SHALL be followed to unbind it to a WFN is specified in pseudocode below. The top-level unbinding function, `unbind_fs`, is called with the formatted string to be unbound as its only argument. The pseudocode references the defined operations on WFNs (cf. 5.4) as well as a number of helper functions also defined in pseudocode. Section 6.2.3.1 summarizes the algorithm in prose. Section 6.2.3.2 provides the pseudocode for the algorithm. Section 6.2.3.3 provides examples of unbinding formatted strings to WFNs.

It is OPTIONAL for implementations of formatted string unbinding to follow the pseudocode, processes, and other non-normative information provided in this section. However, the result of an implementation performing a formatted string unbinding MUST produce the same result as the suggested pseudocode, processes, etc. would produce.

6.2.3.1 Summary of algorithm

Unbinding a formatted string is straightforward, since the attribute values are encoded explicitly and in a fixed left-to-right order in the binding, delimited by colons. (Because a colon may appear embedded in a value string if preceded by the escape character, the parsing function needs to ignore escaped colons.) The algorithm parses the eleven fields of the formatted string, then unbinds each string result. If a field contains only an asterisk, it is unbound to the logical value ANY. If a field contains only a hyphen, it is unbound to the logical value NA. Quoting of non-alphanumeric characters is restored as needed, but the two special characters (asterisk and question mark) are permitted to appear without a preceding escape character. The unbinding procedure performs limited error checking, only ensuring that unquoted special characters adhere to syntax requirements specified in Figure 5-1.

6.2.3.2 Pseudocode for algorithm

```
function unbind_fs(fs)
  ;; Top-level function to unbind a formatted string fs to a wfn.
  result := new().
  ;; NB: the cpe scheme is the 0th component, the cpe version is the
  ;; 1st. So we start parsing at the 2nd component.
  for a = 2 to 12
    do
      v := get_comp_fs(fs,a).    ; get the a'th field string
      v := unbind_value_fs(v).   ; unbind the string
      ;; set the value of the corresponding attribute.
      case a:
        2: result := set(result,part,v).
        3: result := set(result,vendor,v).
        4: result := set(result,product,v).
        5: result := set(result,version,v).
        6: result := set(result,update,v).
        7: result := set(result,edition,v).
        8: result := set(result,language,v).
        9: result := set(result,sw_edition,v).
       10: result := set(result,target_sw,v).
       11: result := set(result,target_hw,v).
       12: result := set(result,other,v).
```

```
      end.
  end.
  return result.
end.

function get_comp_fs(fs,i)
  ;; Return the i'th field of the formatted string.  If i=0,
  ;; return the string to the left of the first forward slash.
  ;; The colon is the field delimiter unless prefixed by a
  ;; backslash.
  ;; For example, given the formatted string:
  ;; cpe:2.3:a:foo:bar\:mumble:1.0:*:…
  ;; get_comp_fs(fs,0) = "cpe"
  ;; get_comp_fs(fs,1) = "2.3"
  ;; get_comp_fs(fs,2) = "a"
  ;; get_comp_fs(fs,3) = "foo"
  ;; get_comp_fs(fs,4) = "bar\:mumble"
  ;; get_comp_fs(fs,5) = "1.0"
  ;; etc.
end.

function unbind_value_fs(s)
  ;; Takes a string value s and returns the appropriate logical
  ;; value if s is the bound form of a logical value. If s is some
  ;; general value string, add quoting of non-alphanumerics as
  ;; needed.
  case s:
    "*": return ANY.
    "-": return NA.
    else:
      ;; add quoting to any unquoted non-alphanumeric characters,
      ;; but leave the two special characters alone, as they may
      ;; appear quoted or unquoted.
      return add_quoting(s).
  end.
end.

function add_quoting(s)
  ;; Inspect each character in string s. Copy quoted characters,
  ;; with their escaping, into the result. Look for unquoted non
  ;; alphanumerics and if not "*" or "?", add escaping.
  result := "".
  idx := 0.
  embedded := false.
  while (idx < strlen(s))
    do
      c := substr(s,idx,idx).  ; get the idx'th character of s.
      if (is_alphanum(c) or c = "_") then
        ;; Alphanumeric characters pass untouched.
        result := strcat(result,c).
        idx := idx + 1.
        embedded := true.
```

```
        continue.
      endif.
      if c = "\" then
        ;; Anything quoted in the bound string stays quoted
        ;; in the unbound string.
        result := strcat(result,substr(s,idx,idx+1)).
        idx := idx + 2.
        embedded := true.
        continue.
      endif.
      if (c = "*") then
        ;; An unquoted asterisk must appear at the beginning or
        ;; end of the string.
        if (idx = 0 or idx = (strlen(s)-1)) then
          result := strcat(result,c).
          idx := idx + 1.
          embedded := true.
          continue.
        else error.
      endif.
      if (c = "?") then
        ;; An unquoted question mark must appear at the beginning or
        ;; end of the string, or in a leading or trailing sequence.
        if ;; ? legal at beginning or end
           (((idx = 0) or (idx = (strlen(s)-1))) or
            ;; embedded is false, so must be preceded by ?
            (!embedded and (substr(s,idx-1,idx-1) = "?")) or
            ;; embedded is true, so must be followed by ?
            (embedded and (substr(s,idx+1,idx+1) = "?"))) then
          result := strcat(result,c).
          idx := idx + 1.
          embedded := false.
          continue.
        else error.
      endif.
      ;; all other characters must be quoted
      result := strcat(result,"\",c).
      idx := idx + 1.
      embedded := true.
  end.
  return result.
end.
```

6.2.3.3 Examples of unbinding a formatted string to a WFN

This section provides a number of examples illustrating the results of unbinding a formatted string to a WFN.

6.2.3.3.1 Example 1

FS: `cpe:2.3:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:*`

Unbinds to this WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update="beta",edition=ANY,language=ANY,
sw_edition=ANY,target_sw=ANY,target_hw=ANY,other=ANY]
```

Notice how the periods in the version string are quoted in the WFN, and all the asterisks are unbound to the logical value ANY.

6.2.3.3.2 Example 2

FS: `cpe:2.3:a:microsoft:internet_explorer:8.*:sp?:*:*:*:*:*:*`

Unbinds to this WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.*",update="sp?",edition=ANY,language=ANY,
sw_edition=ANY,target_sw=ANY,target_hw=ANY,other=ANY]
```

Note how the embedded special characters are unbound untouched in the WFN.

6.2.3.3.3 Example 3

FS: `cpe:2.3:a:hp:insight_diagnostics:7.4.0.1570:-:*:*:online:win2003:x64:*`

Unbinds to this WFN:

```
wfn:[part="a",vendor="hp",product="insight_diagnostics",
version="7\.4\.0\.1570",update=NA,edition=ANY,language=ANY,
sw_edition="online",target_sw="win2003",target_hw="x64",
other=ANY]
```

Note how the lone hyphen in the update field unbinds to the logical value NA, and how the lone asterisks unbind to the logical value ANY.

Contrast the above example with the modified version below:

FS: `cpe:2.3:a:hp:insight_diagnostics:7.4.*.1570:*:*:*:*:*:*`

This raises an error during unbinding, due to the embedded unquoted asterisk in the version attribute.

6.2.3.3.4 Example 4

FS: `cpe:2.3:a:foo\\bar:big\$money:2010:*:*:*:special:ipod_touch:80gb:*`

Unbinds to this WFN:

```
wfn:[part="a",vendor="foo\\bar",product="big\$money",
version="2010",update=ANY,edition=ANY,language=ANY,
sw_edition="special",target_sw="ipod_touch",target_hw="80gb",
other=ANY]
```

Note how the quoted special characters retain their quoting in the WFN.

7. Bound Name Conversions

This section specifies the procedures that SHOULD be followed when converting between the two required bound forms of WFNs. Section 7.1 specifies the procedure for converting a URI binding to a formatted string binding, and Section 7.2 specifies the inverse conversion. Both conversions are *round-trip safe*, meaning that taking one of the two bound forms, converting it to the other form, then converting that back to the original binding form, will result in the original binding.

7.1 Converting a URI to a Formatted String

Given a URI *u* which conforms to the CPE v2.2 specification, the procedure for converting it to a formatted string *fs* has two steps:

```
function convert_uri_to_fs(u)
  w := unbind_uri(u).
  fs := bind_to_fs(w).
  return fs.
end.
```

7.2 Converting a Formatted String to a URI

Given a formatted string *fs* which conforms to the description in Section 6.2.2, the procedure for converting it to a URI has two steps:

```
function convert_fs_to_uri(fs)
  w := unbind_fs(fs).
  uri := bind_to_uri(w).
  return uri.
end.
```

Appendix A—References

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

A.1 Normative References

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

[RFC2119] Bradner, S. (1997). *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. See http://www.ietf.org/rfc/rfc2119.txt.

[RFC2234] Crocker, D. and P. Overell. (1997). *Augmented BNF for Syntax Specifications: ABNF*. Internet RFC 2234, November 1997. See http://www.ietf.org/rfc/rfc2234.txt.

[RFC3629] Yergeau, F. (2003). *UTF-8, a transformation format of ISO 10646.* Internet RFC 3629, November 2003. See http://www.ietf.org/rfc/rfc3629.txt.

[RFC3986] Berners-Lee, T., Fielding, R. and L. Masinger. (2005). *Uniform Resource Identifier (URI): Generic Syntax.* Internet RFC 3986, January 2005. See http://www.ietf.org/rfc/rfc3986.txt.

[RFC5646] Phillips, A. and M. Davis. (2009). *Tags for Identifying Languages.* RFC 5646, September 2009. See http://www.ietf.org/rfc/rfc5646.txt.

A.2 Informative References

[ISO19770-2] ISO/IEC 19770-2. (2009). *Software Identification Tag.* November 2009. See http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53670.

[Silberschatz] Silberschatz, A., Peterson, J. L., and Galvin, P. B. *Operating System Concepts—Third Edition*. Reading, MA: Addison-Wesley, 1991.

[SP800-117] Quinn, S., Scarfone, K., Barrett, M., and Johnson, C., NIST Special Publication 800-117, *Guide to Adopting and Using the Security Content Automation Protocol*, July 2010. See http://csrc.nist.gov/publications/PubsSPs.html#800-117.

[SP800-126] Waltermire, D., Quinn, S., Scarfone, K., and Halbardier, A., NIST Special Publication 800-126 Revision 2, *The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.2*, July 2011. See http://csrc.nist.gov/publications/PubsSPs.html#800-126.

[TUCA] *Common Platform Enumeration (CPE) Technical Use Case Analysis*, The MITRE Corporation, November 2008. See http://cpe.mitre.org/about/use_cases.html.

Appendix B—Change Log

Release 0 – 9 June 2010

- Initial draft specification released to the CPE community as a read ahead for the CPE Developer Days Workshop.

Release 1 – 23 June 2010

- Minor edits to audience description.
- Minor editorial changes throughout the document.
- In section on Conformance, added a requirement that claims of conformance be made explicit in product documentation. Modified the third clause to allow conformers to "produce **and/or** consume", that is, an "and" became an "and/or", since some applications only need to produce and others only need to consume. Relaxed the requirement to consume legacy CPE names from a MUST to a SHOULD, since some applications may have no need to consume legacy content.
- Added an ABNF grammar to define character strings permitted as attribute values in WFNs.
- Switched to using the words/phrases "to quote" and "quoting" in place of "to escape" and "escaping" when referring to use of the escape character, to be more consistent with standard regular expression usage.
- Removed all mention of and support for the logical value UNKNOWN.
- Clarified the view that the logical value NA should also be used if an attribute value is assessed to be null.

Release 2 – 22 April 2011

- Reorganized the sequence of several sections and sub-sections.
- Narrowed the specification of the permitted character set for attribute-value strings.
- Restricted the use of special characters to the beginning and/or end of attribute-value strings.
- Clarified the specification of percent-encoding in the URI binding.
- Changed the prefix of the formatted string binding from "cpe23:" to "cpe:2.3:".
- Added support for binding special characters into the URI by re-purposing the otherwise-proscribed "%00" and "%01" percent-encoding forms.
- Revised pseudocode to support new features (percent encoding and decoding, binding and unbinding of special characters).
- Ensured that uppercase letters are transformed to lowercase in unbind_URI.
- Added and revised examples for consistency.
- Minor editorial changes throughout the document.
- Added a reference to a new XML schema that defines the acceptable CPE bound name formats.

Release 3 – 11 July 2011

- Clarified guidelines on use of legacy edition attribute.
- Bugfixes to pseudocode for binding and unbinding.
- Changed the “%00” and “%01” forms to “%01” and “%02”, respectively, in response to community feedback.

Release 4 – 30 August 2011

- Final release of CPE Naming 2.3 specification.
- Made minor editorial changes.
- Clarified requirements involving attribute-specific valid-values lists.
- Explained that binding and unbinding implementations must achieve the same results as the provided pseudocode, processes, etc., but do not have to implement the pseudocode, processes, etc.