NIST Interagency Report 7696

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

Common Platform Enumeration: Name Matching Specification Version 2.3

Mary C. Parmelee
Harold Booth
David Waltermire
Karen Scarfone

NIST Interagency Report 7696

Common Platform Enumeration: Name Matching Specification Version 2.3

Mary C. Parmelee
Harold Booth
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 7696
28 pages (Aug. 2011)

Acknowledgments

The authors, Mary C. Parmelee of The MITRE Corporation, Harold Booth and 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 Brant A. Cheikes of The MITRE Corporation, Paul Cichonski of NIST, Adam Halbardier of Booz Allen Hamilton, Seth Hanford of Cisco Systems, Inc., Tim Keanini of nCircle, Kent Landfield of McAfee, Inc., 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) Name Matching version 2.3 specification. The CPE Name Matching specification is part of a stack of CPE specifications that support a variety of use cases relating to IT product description and naming. The CPE Name Matching specification provides a method for conducting a one-to-one comparison of a source CPE name to a target CPE name. In addition to defining the specification, this report also defines and explains the requirements that IT products must meet for conformance with the CPE Name Matching 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 in 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 Name Matching is one of several modular CPE specifications that work together in layers to perform various functions. CPE Name Matching defines a method for conducting a one-to-one comparison of a source CPE name to a target CPE name. By logically comparing CPE names as sets of values, CPE Name Matching methods can determine if common set relations hold. For example, CPE Name Matching can determine if the source and target names are equal, if one of the names is a subset of the other, or if the names are disjoint.

One example of the value of CPE Name Matching is in determining if a particular product is installed on a system. Suppose that an organization is identifying which of its systems have any variation of Microsoft Internet Explorer 8 installed. This could be represented with the following well-formed CPE name (WFN) [CPE23-N:5.1]:

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

An asset management tool could collect information on the software installed on a system and compare its Internet Explorer installation characteristics to the WFN above. Suppose that the WFN for a particular installed instance of Internet Explorer was reported as:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update=NA,edition=NA,language="en\-us"]
```

Using these two example WFNs, CPE Name Matching methods perform a pairwise comparison of attribute values in the first (source) WFN to those in the second (target) WFN, yielding a list of the set relations between each pair of attributes (e.g., equal, superset). This list of comparison results is then assessed, leading to a determination that the first WFN represents a superset of the second WFN. This can be interpreted to mean that the system being examined does indeed have a variation of Microsoft Internet Explorer 8 installed.

Although this may seem like a trivial example, CPE Name Matching is a powerful and flexible way of performing product comparisons in a standardized, automated manner. CPE Name Matching is also used by other CPE specifications to conduct more complex tasks, such as searching for product names in CPE dictionaries and performing complex comparisons of sets of product versions—for example, determining if a system is running a particular operating system version, running two particular applications, and not running a third particular application.

1.1 Purpose and Scope

This report defines the specification for CPE Name Matching version 2.3. The report also defines and explains the requirements that producers of CPE Name Matching implementations, such as software and services, must meet to claim conformance with version 2.3 of the CPE Name Matching specification.

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

1.2 Audience

This report is intended for two main audiences: IT management tool developers and the authors and editors of CPE content. Readers of this report should already be familiar with CPE naming concepts and conventions as defined in [CPE23-N].

1.3 Document Structure

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

- Section 2 defines the key 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 describes the foundational concepts, name constructs, and technical constraints associated with CPE name matching.
- Section 6 defines CPE set relations.
- Section 7 describes expected name matching behavior in pseudocode.
- Appendix A lists normative and informative references.
- Appendix B describes how to implement CPE 2.2 equivalent matching capabilities using CPE 2.3 matching functions.
- Appendix C provides a change log that documents significant changes to major drafts of the specification.

1.4 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 [RFC 2119].

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 of this document. 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., [CPE23-N:7.2] is a citation for the CPE 2.3 Naming specification, Section 7.2)
- 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. 3.1) is a citation for Section 3.1 of this document).

The phrase “CPE Name Matching” is capitalized only when referring to the proper name of this specification. When “CPE name matching” is otherwise used, such as a verb phrase, only the CPE acronym is capitalized.

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. Whenever possible, we reference semantically equivalent functions from 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 selected terms and acronyms used within the document.

2.1 Definitions

The following definitions apply to the CPE Name Matching specification. These definitions build on the definitions from the CPE Naming specification [CPE23-N]. They have been adapted where practical from authoritative sources, such as industry, national, and international standard specifications.

Attribute: A property or characteristic of a computing product. CPE 2.2 commonly used the term "component" instead of "attribute". CPE 2.3 uses the term "attribute" to clarify the distinction between CPE 2.2 name "components" and computing components, such as software modules. Examples of CPE 2.3 attributes are part, vendor, product, and version. CPE attributes and their value constraints are defined in the CPE Naming specification [CPE23-N:5.2, 5.3].

Attribute-Value (A-V) Pair: A tuple a=v in which a (the attribute) is an alphanumeric label representing a property or state, and v (the value) is the value assigned to the attribute.

CPE Attribute Comparison: The first phase of CPE name matching, where a matching engine compares each of the A-V pairs of a source CPE name to the corresponding A-V pair of a target name in order to specify one of four possible logical attribute comparison relations for each attribute in a CPE name.

CPE Name Comparison: The second phase of CPE name matching, where the individual attribute comparison results from the first phase are analyzed as a collection to determine an overall comparison result for the two names. The result of a name comparison is the identification of the relationship between a source CPE name and target CPE name.

CPE Name Matching: A one-to-one source-to-target comparison of CPE names. CPE name matching has two phases: attribute comparison and name comparison. CPE name matching compares source-to-target attribute values at the attribute comparison level, and then applies rules to the set of attribute relations to determine a name match, such as the names being equal or the source name being a superset of the target name.

Extension: The set of individual products to which a WFN refers.

Quote: To precede printable non-alphanumeric characters (e.g., *, $, ?) with the backslash (\) escape character in a value string. When a non-alphanumeric character is quoted in a WFN, it SHALL be processed as string data. When a non-alphanumeric character is unquoted in a WFN, it may be interpreted as a special character by CPE 2.3 specifications, including this one.

Source Name: A single WFN that a matching engine compares to a target WFN to determine whether or not there is a source-to-target match. (This is the X value in the CPE 2.2 matching algorithm.)

Source Value: A single value that a matching engine compares to a corresponding target value to determine whether or not there is a source-to-target match. Source values include A-V pairs or set relation values (e.g., superset or subset).

Special Character: A non-alphanumeric character that may be defined by one or more CPE specifications to have a special meaning when it appears unquoted in a WFN. Special characters typically trigger a processor to perform a given function. The rules for escaping CPE special characters are defined in the CPE Naming specification [CPE23-N:5.3.2].

Target Name: A single WFN that is the target of a matching process. A matching engine compares a source WFN to a target WFN to determine whether or not there is a source-to-target match. In CPE 2.2 terms a target name is a single item in the list of known values (each N of K) and is equivalent to the N value in the CPE 2.2 Matching algorithm.

Target Value: A single value that is the target of a matching process. A matching engine compares a source value to a target value to determine whether or not there is a source-to-target match. Source values include AV pairs or set relation values (e.g., superset or subset).

Value String: A value assigned to an attribute of a WFN. It must be a non-empty contiguous string of bytes encoded using printable Unicode Transformation Format-8 (UTF-8) characters with hexadecimal values between x00 and x7F.

Well-Formed CPE Name (WFN): A logical construct that constitutes an unordered list of A-V pairs that collectively describe or identify one or more operating systems, software applications, or hardware devices. Unordered means that there is no prescribed order in which A-V pairs should be listed, and there is no specified relationship (hierarchical, set-theoretic, or otherwise) among attributes. WFNs must satisfy the criteria specified in the CPE Naming specification [CPE23-N:5.2]. For a full description and usage constraints on WFN logical attribute values, see Section 5 of the CPE Naming specification [CPE23-N:5].

2.2 Abbreviations

A-V	Attribute-Value Pair
CPE	Common Platform Enumeration
GNU	GNU's Not Unix (recursive acronym)
IR	Interagency Report
IT	Information Technology
ITL	Information Technology Laboratory
NIST	National Institute of Standards and Technology
RFC	Request for Comment
SCAP	Security Content Automation Protocol
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 has 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 Name Matching specification builds on the foundation of the CPE Naming specification [CPE23-N]. Many of the concepts and methods that are applied in this specification are defined in the CPE Naming specification. Specifications layered above the CPE Name Matching specification in the CPE stack build on the matching criteria and logic defined in this document.

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.

The primary changes in CPE Name Matching since CPE 2.2 are:

1. The "prefix property" [CPE22:4] was eliminated from CPE 2.3.
2. New matching criteria and logic support a broad range of use cases.
3. CPE name matching is defined in terms of a WFN so that it can be implemented in a common way regardless of how names are bound for machine processing and interchange.
4. Attribute matching supports single-character and multi-character wild card values.

4. Conformance

Product manufacturers 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 ensure the interoperability of conformant implementations through common CPE name matching functionality.

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. An implementation MUST implement the behavior that is specified in the pseudocode of this document (cf. 7).
3. An implementation MUST produce the identical results for CPE attribute and name comparison relations that are specified in this document (cf. 5, 6).
4. An implementation MUST satisfy the technical constraints as defined in this document (cf. 5.2)

5. Name Matching Overview

The CPE Name Matching specification interprets the Well-Formed CPE Name (WFN) as a set-theoretic construct. That is, a WFN is treated as an expression that refers to a set of individual products having certain attribute values. The set of individual products to which a WFN refers is called the *extension* of the WFN. For example, the extension of the WFN

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

is the set of individual products with the `part` attribute equal to `"a"` (i.e., applications), the `vendor` attribute equal to `"microsoft"`, the `product` attribute equal to `"internet_explorer"`, the `version` attribute equal to `"8\.0"`, and all the other (unspecified) attribute values with any value. It is important to keep in mind that the attribute values serve as constraints on membership in the WFN's extension; they are ANDed together because, to be a member of a WFN's extension, an individual product must satisfy all the attribute values.

This conception of WFNs permits an intuitive formulation of WFN matching as the assessment of the set relationship (e.g., subset, superset) between the two extensions of the WFNs being compared. This specification builds on the set-theoretic conception of WFNs to define common matching functionality for the assessment of the set relation between a source WFN and a target WFN. The functionality described in this specification encompasses two main parts: a method for comparing attribute values within the source and target WFNs to identify their set relations, and a method for comparing source and target WFNs as sets of attribute relations. Taken together, these two parts provide for basic tool interoperability, while remaining flexible and extensible enough to apply to a broad range of use cases.

The CPE v2.2 specification [CPE22] provided a single definition of what it means for a source CPE name to "match" a target name. In contrast, this specification takes a more open and flexible approach. Here, CPE name matching is defined to return individual results for each pairwise attribute comparison, along with a single overall set-theoretic result for a name comparison. CPE's developers have chosen not to define a single notion of "name match" because experience has shown that name matching distinctions are often use-case dependent. For example, when a source WFN is generated from the sparse results of a non-authenticated asset inventory tool, matching of only one or two CPE attribute values may constitute a meaningful "name match" for some applications. In contrast, when both the source and target WFNs are fully specified, common names in an authoritative CPE dictionary, it may be reasonable to decide that a "name match" requires an exact match of all CPE attribute values in both names. In order to remain flexible enough to support these and other use cases, the CPE Name Matching specification leaves the majority of decisions about what constitutes a "name match" to CPE implementers at design time. This flexibility is expected to free the CPE community to develop new and innovative ways to define CPE name matches as required to satisfy various application needs.

Although the following matching capabilities may build on the foundation of the CPE Name Matching specification, they are outside of its scope:

1. Multiple name results. Although CPE Name Matching methods can be sequentially applied to a list of target names, they return only the first match found in the list. CPE Name Matching methods do not return lists of results.
2. Many-to-many comparisons. When comparing a list of source names to a list of target names, the CPE Name Matching specification provides a foundation from which to build list-to-list comparisons, but it does not define many-to-many comparisons.

3. Weighting of matching results. CPE Name Matching does not specify how to determine whether or not one match is more relevant than another. For example, the algorithm does not distinguish whether a match of a `version` attribute value is more or less relevant than a match of a `language` attribute value.

The remainder of this section defines the foundational CPE name constructs and technical constraints associated with CPE name matching.

5.1 CPE Name Constructs

CPE name matching is defined to be independent of any bound form of a CPE name. Rather, it is defined only in terms of the logical constructs of a WFN. The possible attribute values of a WFN include the logical values ANY and NA, value strings, and special characters. For a full description and basic usage constraints on WFN attribute values, see the CPE Naming specification [CPE23-N:5].

The CPE Naming specification designates the asterisk (*) and the question mark (?) as special characters for use in the CPE attribute-value strings of a WFN. When these characters appear unquoted within a CPE attribute-value string, they may be interpreted as having a special meaning by CPE specifications [CPE23-N:5.3.2]. The CPE Name Matching specification assigns special interpretations to these unquoted characters. For matching purposes, an unquoted asterisk in an attribute-value string SHALL be interpreted as a multi-character wild card, and an unquoted question mark SHALL be interpreted as a single-character wild card. Logically, these wild cards translate to multi-character ANY and single-character ANY, respectively.

Although the CPE Name Matching specification is defined in terms of WFNs, CPE Name Matching implementations are NOT REQUIRED to transform CPE names into WFNs prior to matching. In practice, implementers MAY choose to unbind CPE names to WFNs prior to matching, or they MAY apply matching to the bound form of their choice.

When applying matching directly to bound forms (e.g., URIs or formatted strings), implementers should be aware of the ways in which unquoted special characters are encoded [CPE23-N:6.1.2.1.2]. In URI bindings, the unquoted question mark is encoded as the character sequence `%01`, and the unquoted asterisk is encoded as the character sequence `%02`. In formatted string bindings, the unquoted question mark is encoded as a question mark without a preceding escape (backslash) character, and the unquoted asterisk is encoded as an asterisk without a preceding escape character.

5.2 Technical Constraints

This specification places the following constraints on usage of the unquoted question mark in attribute-value strings:

1. An unquoted question mark MAY be used at the beginning and/or the end of an attribute-value string. Examples: `"?foo"`, `"bar?"`, `"?baz?"`
2. A contiguous sequence of unquoted question marks MAY appear at the beginning and/or the end of an attribute-value string. Examples: `"??foo"`, `"bar???"`, `"??baz???"`
3. An unquoted question mark SHALL NOT be used in any other place in an attribute-value string. Examples of illegal usage: `"foo?bar"`, `"bar??baz"`, `"q??x"`

This specification places the following constraints on usage of the unquoted asterisk in attribute-value strings:

1. A single unquoted asterisk MAY be used as the entire attribute-value string. Example: `"*"`
2. A single unquoted asterisk MAY be used at the beginning and/or end of an attribute-value string. Examples: `"*foo"`, `"bar*"`, `"*baz*"`
3. An unquoted asterisk SHALL NOT be used in any other place in an attribute-value string. Example of illegal usage: `"foo*bar"`, `"**foo"`, `"bar***"`

Unquoted question marks and asterisks MAY appear in the same attribute-value string as long as they meet the constraints above. Examples of legal usage: `"?foo*"`, `"*bar??"`. Examples of illegal usage: `"*?foobar"`, `"foobar*?"`.

CPE Name Matching implementations MUST satisfy the following technical constraints:

1. The logical meaning that is applied to the unquoted characters asterisk (*) and question mark (?) MUST be applied as defined in this specification.
2. Attribute comparison MUST be performed prior to name comparison.
3. The collective relations of the attribute comparison of a source CPE name to a target CPE name SHALL be used as input to name comparison.
4. Final CPE name matching results SHALL provide matching results for each attribute comparison in a CPE name as well as an overall name comparison result.

6. Set Relations

This section defines CPE Name Matching set relations and describes how they are applied to compare CPE attributes and names.

There are four possible set relations between a source WFN and a target WFN. Although the contents of source and target and the applied logic vary among name matching processes, the meaning of the relation itself remains consistent throughout this specification. Table 6-1 defines the notation and meaning of each set relation.

Table 6-1: CPE Name Matching Set Relations

No.	Notation	Definition
1	⊃	The source is a SUPERSET of the target
2	⊂	The source is a SUBSET of the target
3	=	The source and target are EQUAL
4	≠	The source and target are mutually exclusive or DISJOINT

The SUPERSET and SUBSET relations defined in this specification are the conventional, “non-proper” set relations: that is, a set is not only EQUAL to itself, but it is also a SUPERSET and a SUBSET of itself.

Given the set-theoretic conception of WFNs, WFN extensions are sets containing zero or more individuals. Each A-V pair of a WFN can thus be thought of as a subset of the WFN's full extension, namely, the subset of individuals from the full extension having the particular attribute value. This idea allows us to design source-to-target WFN comparison as a two-stage process: a sequence of pairwise A-V comparisons, yielding a list of results, followed by a holistic evaluation of the results list to arrive at an overall determination of the set-theoretic relationship between source and target. The next section describes the attribute comparison relations.

6.1 Attribute Comparison Relations

The first of two CPE name matching phases, the attribute comparison process compares each A-V pair in a source name to its corresponding A-V pair in a target name, matching it to one of the four possible set relations. For example, comparing the source A-V `vendor=ANY` to the target A-V `vendor="microsoft"` matches to set relation number 1 in Table 6-1 (⊃), where the source A-V represents a superset of the target A-V.

Table 6-2 enumerates all combinations of CPE WFN A-Vs and defines the set relation for each comparison. The following key describes the attribute value notation for Table 6-2.

1. ANY and NA are logical values as defined in [CPE23-N:5.3.1]
2. i is a wildcard-free attribute-value string, e.g., "foo"
3. k is a wildcard-free attribute-value string that is not identical to i, e.g., "bar"
4. m + wild cards is an attribute-value string containing a legal combination of unquoted question mark or asterisk wild cards at the beginning and/or the end of the string, e.g., "*b??"

Table 6-2: Enumeration of Attribute Comparison Set Relations

No.	Source A-V	Target A-V	Relation
1	ANY	ANY	=
2	ANY	NA	⊃
3	ANY	i	⊃
4	ANY	m + wild cards	undefined
5	NA	ANY	⊂
6	NA	NA	=
7	NA	i	≠
8	NA	m + wild cards	undefined
9	i	i	=
10	i	k	≠
11	i	m + wild cards	undefined
12	i	NA	≠
13	i	ANY	⊂
14	m_1 + wild cards	m_2	⊃ or ≠
15	m + wild cards	ANY	⊂
16	m + wild cards	NA	≠
17	m_1 + wild cards	m_2 + wild cards	undefined

When comparing string literals, matching results MUST be insensitive to lexical case.

Wild card usage is a new OPTIONAL feature in CPE 2.3. Wild cards MAY be included as part of the source value (lines 14-17 of Table 6-2). Wild cards SHOULD NOT be included in the target value—in this specification the inclusion of wild card characters in a target value yields an undefined result. See Section 6.3 of this document for a description of wild card matching criteria and relations.

In line 14 of Table 6-2, the result of the comparison depends on whether the wildcard comparison returns a positive string match. For example, if the source value is "9\.*" and the target value is "9\.3", then the result of the comparison is ⊃ (SUPERSET). If, however, the source value is "9\.*" and the target value is "8\.3", then the string match fails and the result of the comparison is ≠ (DISJOINT).

The attribute comparison phase results in a list of relations, one for each pairwise comparison of WFN attributes. Table 6-3 illustrates a set of source and target A-Vs and the resulting set of attribute comparison relations.

Table 6-3: Attribute Comparison Example

Attribute	Part	Vendor	Product	Version	Update	Edition
Source Value	a	Adobe	ANY	9.*	ANY	PalmOS
Target Value	a	ANY	Reader	9.3.2	NA	NA
Relation Set	=	⊂	⊃	⊃	⊃	≠

6.2 Name Comparison Relations

The second of two CPE name matching phases, the name comparison process evaluates the attribute relations in the relation set resulting from the attribute comparison phase of the matching process. The results of a CPE name comparison is a single set relation between the source and target WFNs. In keeping with the goal to remain flexible enough to support a broad range of use cases, while specifying enough commonality to ensure basic interoperability between CPE Name Matching conformant tools, CPE 2.3 provides four basic name comparison relations. These relations were chosen for two reasons: they are the least likely to be use-case dependent, and they provide functional backward compatibility with the CPE 2.2 name matching process.

The four name comparison relations are described in Table 6-4.

Table 6-4: Required CPE Name Comparison Relations

No.	If Attribute Relation Set =	Then Name Comparison Relation
1	If any attribute relation is DISJOINT (≠)	Then CPE name relation is DISJOINT (≠)
2	If all attribute relations are EQUAL (=)	Then CPE name relation is EQUAL (=)
3	If all attribute relations are SUBSET (⊂) or EQUAL (=)	Then CPE name relation is SUBSET(⊂)
4	If all attribute relations are SUPERSET (⊃) or EQUAL (=)	Then CPE name relation is SUPERSET (⊃)

Although additional name matching relations MAY be defined by other CPE specifications or by the CPE community to meet their operational needs, these four name comparison relations are the minimal REQUIRED set for baseline interoperability of CPE Name Matching implementations. Four corresponding name comparison functions define the expected behavior for CPE name comparison in the pseudocode specification in Section 7.2. CPE implementers who wish to emulate the functionality of the CPE 2.2 Matching algorithm should note that name comparison numbers 1 and 3 in Table 6-4 are equivalent to a final result of FALSE in CPE 2.2, while name comparison numbers 2 and 4 are equivalent to a CPE 2.2 result of TRUE. See Appendix B for a discussion of how to implement CPE 2.2 equivalent matching capabilities using CPE 2.3 name matching procedures.

6.3 Wild Card Attribute Comparison

A wild card (i.e., an unquoted special character) MAY be included as part of the source value when performing attribute value comparisons. The unquoted asterisk (*) character is a wild card for zero or more characters in the target value, and the unquoted question mark (?) character is a wild card for zero or one characters in the target value. Following [CPE23-N], wild card characters SHALL BE restricted to

appearing at the beginning and/or the end of strings. Use of wild cards embedded within a string is not supported.

When comparing a source string containing wild cards to a target string (cf. line 14 in Table 6-2), one of two set relation results are possible: SUPERSET (⊃) or DISJOINT (≠). If the source string (treated as a simple regular expression) matches the target string, the set-theoretic relation is SUPERSET (⊃), because the set denoted by the target string includes or is equal to the set denoted by the target. If the source string (treated as a simple regular expression) fails to match the target, the set-theoretic relation is DISJOINT (≠), because the set denoted by the source does not overlap the set denoted by the target.

7. CPE Name Matching Pseudocode

This section specifies the required common matching capabilities in terms of an abstract pseudocode programming language. The input/output behavior of all functions defined in pseudocode should be considered normative. The pseudocode implementations themselves should be considered informative, as the algorithms are written for clarity and simplicity rather than for efficiency.

7.1 Overview of CPE Name Matching Pseudocode

The following core functions comprise the required common matching capabilities:

- **`CPE_DISJOINT`**: This function compares two WFNs and returns TRUE if the set-theoretic relation between the names is DISJOINT.
- **`CPE_EQUAL`**: This function compares two WFNs and returns TRUE if the set-theoretic relation between the names is EQUAL.
- **`CPE_SUBSET`**: This function compares two WFNs and returns TRUE if the set-theoretic relation between the names is (non-proper) SUBSET.
- **`CPE_SUPERSET`**: This function compares two WFNs and returns TRUE if the set-theoretic relation between the names is (non-proper) SUPERSET.
- **`Compare_WFNs`**: This function compares two WFNs and returns a list of pairwise attribute-value comparison results. This function is required by the functions listed above. It provides full access to the individual comparison results to enable use-case specific implementations of novel name-comparison algorithms.

These core functions depend on a number of support functions, including `compare()`, `compareStrings()`, and several string-processing functions. Section 7.2 defines the core functions, and Section 7.3 defines the support functions.

7.2 CPE Name Matching Pseudocode: Core Functions

The following CPE Name Matching pseudocode defines the required core matching functions.

```
;; Begin CPE DISJOINT function. Input arguments are WFNs.
;; Returns TRUE if the set relation between source and target is DISJOINT,
;; otherwise FALSE.
function CPE_DISJOINT(source, target)
  result_list := Compare_WFNs(source, target).
  ;; If any pairwise comparison returned DISJOINT (≠) then the overall
  ;; name relationship is DISJOINT (≠).
  foreach result in result_list do
    if (result = DISJOINT (≠)) then return TRUE.
  end.
  return FALSE.
end.

;; Begin CPE EQUAL function. Input arguments are WFNs.
;; Returns TRUE if the set relation between source and target is EQUAL,
;; otherwise FALSE.
function CPE_EQUAL(source, target)
  result_list := Compare_WFNs(source, target).
  ;; If every pairwise comparison returned EQUAL (=) then the overall
  ;; name relationship is EQUAL (=).
  foreach result in result_list do
```

```
    if (result != EQUAL (=)) then return FALSE.
  end.
  return TRUE.
end.

;; Begin CPE SUBSET function. Input arguments are WFNs.
;; Returns TRUE if the set relation between source and target is SUBSET,
;; otherwise FALSE.
function CPE_SUBSET(source, target)
  result_list := Compare_WFNs(source, target).
  ;; If any pairwise comparison returned something other than SUBSET or
  ;; EQUAL, then SUBSET is false.
  foreach result in result_list do
    if ((result != SUBSET (⊂)) and (result != EQUAL (=))) then
      return FALSE.
  end.
  return TRUE.
end.

;; Begin CPE SUPERSET function. Input arguments are WFNs.
;; Returns TRUE if the set relation between source and target is
;; SUPERSET, otherwise FALSE.
function CPE_SUPERSET(source, target)
  result_list := Compare_WFNs(source, target).
  ;; If any pairwise comparison returned something other than SUPERSET
  ;; or EQUAL, then SUPERSET is false.
  foreach result in result_list do
    if ((result != SUPERSET (⊃)) and (result != EQUAL (=))) then
      return FALSE.
  end.
  return TRUE.
end.

;; Compare each attribute of the Source WFN to the Target WFN.
;; Inputs to the function are WFNs.
;; Output is a list of pairwise attribute comparison results.
function Compare_WFNs(source, target)
  ;; Create a new associative array table.
  result := new table.
  ;; Compare results using the get() function defined in Section 5.4.2
  ;; of the CPE Naming specification.
  put(result, part, compare(get(source, part), get(target, part))).
  put(result, vendor, compare(get(source, vendor), get(target, vendor))).
  put(result, product, compare(get(source, product), get(target, product))).
  put(result, version, compare(get(source, version), get(target, version))).
  put(result, update, compare(get(source, update), get(target, update))).
  put(result, edition, compare(get(source, edition), get(target, edition))).
  put(result, language, compare(get(source, language), get(target,
    language))).
  put(result, sw_edition, compare(get(source, sw_edition), get(target,
    sw_edition))).
  put(result, target_sw, compare(get(source, target_sw), get(target,
    target_sw))).
  put(result, target_hw, compare(get(source, target_hw), get(target,
    target_hw))).
  put(result, other, compare(get(source, other), get(target, other))).
```

```
  return result.
end.
```

7.3 CPE Name Matching Pseudocode: Support Functions

The following CPE Name Matching pseudocode defines the required matching support functions.

```
;; The compare() function is a support function for Compare_WFNs.
;; Input to the function is a pair of attribute values, which may
;; be logical values (ANY or NA) or string values.
;; Output is the attribute comparison relation as defined in Section 6.1
;; of this document.
function compare(source, target)
  if (is_string(source)) then source := to_lowercase(source).
  if (is_string(target)) then target := to_lowercase(target).
  ;; In this specification, unquoted wildcard characters in the target
  ;; yield an undefined result. Table 6-2, lines 4, 8, 11 and 17.
  if (is_string(target) and contains_wildcards(target)) then
    return UNDEFINED.
  ;; If source and target attribute values are equal, then the
  ;; result is EQUAL (=). Table 6-2, lines 1, 6, 9.
  if (source = target) then return EQUAL (=).
  ;; If source attribute value is ANY, then the result is SUPERSET.
  ;; Table 6-2, lines 2, 3.
  if (source = ANY) then return SUPERSET (⊃).
  ;; If target attribute value is ANY, then the result is SUBSET.
  ;; Table 6-2, lines 5, 13, 15.
  if (target = ANY) then return SUBSET (⊂).
  ;; If either source or target attribute value is NA then the
  ;; result is DISJOINT (≠). Table 6-2, lines 7, 12, 16.
  if (source = NA or target = NA) then return DISJOINT (≠).
  ;; If we get to this point, we are comparing two strings, so call
  ;; compareStrings(). Table 6-2, lines 10, 14.
  return compareStrings(source, target).
end.

;; The compareStrings() function compares a source string to a target
;; string, and addresses the condition in which the source string includes
;; unquoted special characters. It performs a simple regular expression
;; match, with the assumption that (as required) unquoted special characters
;; appear only at the beginning and/or the end of the source string. It
;; also properly differentiates between unquoted and quoted special
;; characters.
function compareStrings(source, target)
  start := 0.
  end := strlen(source).
  begins := 0.
  ends := 0.
  if (substr(source,0,0) = "*") then
    start := 1.
    begins := -1.
  else
    while ((start < strlen(source)) and
           (substr(source,start,start) = "?")) do
      start := start + 1.
```

```
      begins := begins + 1.
    end.
  endif.
  if ((substr(source,end-1,end-1) = "*") and
      (isEvenWildcards(source,end-1))) then
    end := end - 1.
    ends := -1.
  else
    while ((end > 0) and
           (substr(source,end-1,end-1) = "?") and
           (isEvenWildcards(source,end-1))) do
      end := end - 1.
      ends := ends + 1.
    end.
  endif.
  source := substr(source,start,end).
  index := -1.
  leftover := strlen(target).
  while (leftover > 0) do
    index := indexOf(target,source,index+1).
    if (index = -1) then break.
    escapes := countEscapeCharacters(target, 0, index).
    if ((index > 0) and
        (begins != -1) and
        (begins < (index - escapes))) then break.
    escapes := countEscapeCharacters(target, index+1, strlen(target)).
    leftover := strlen(target) - index - escapes - strlen(source).
    if ((leftover > 0) and ((ends != -1) and (leftover > ends)))
      then continue.
    return SUPERSET (⊃).
  end.
  return DISJOINT (≠).
end.

;; Function countEscapeCharacters takes a string str, a start index start,
;; and an end index end. Starting at the start offset into str, it counts
;; and returns the number of distinct escape (backslash) characters found,
;; up to and including the end index.
function countEscapeCharacters(str, start, end)
  result := 0.
  active := FALSE.
  i := 0.
  while (i < end) do
    active := (!active and (substr(str,i,i) = "\")).
    if (active and (i >= start)) then result := result + 1.
    i := i + 1.
  end.
  return result.
end.

;; Function isEvenWildcards returns true if an even number of
;; escape (backslash) characters precede the character at index
;; idx in string str.
function isEvenWildcards(str,idx)
  result := 0.
  while ((idx > 0) and (substr(str,idx-1,idx-1) = "\")) do
```

```
    idx := idx - 1.
    result := result + 1.
  end.
  return isEvenNumber(result).
end.

;; The is_string() function is a support function for compare().
function is_string(arg)
  ;; This function should return TRUE if arg is a string value,
  ;; and FALSE if arg is a logical value (ANY or NA).
end.

;; The contains_wildcards() function is a support function for compare().
function contains_wildcards(string)
  ;; Input to this function is a string value.
  ;; This function should return TRUE if the string contains any unquoted
  ;; special characters (question-mark or asterisk), otherwise FALSE.
  ;; Ex: contains_wildcards("foo") => FALSE
  ;; Ex: contains_wildcards("foo\?") => FALSE
  ;; Ex: contains_wildcards("foo?") => TRUE
  ;; Ex: contains_wildcards("\*bar") => FALSE
  ;; Ex: contains_wildcards("*bar") => TRUE
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 zero being the first character, and ending at the e'th
  ;; character. If b = e, returns the b'th character. b must be <= e.
  ;; Returns nil if b >= strlen(s).
end.

function indexOf(str1,str2,off)
  ;; Searches str1 for the first occurrence of the string str2, starting
  ;; at the offset off into str1. Semantics equivalent to the Java
  ;; string indexOf method. Returns -1 if str2 is not found.
end.

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

[CPE23-N] Cheikes, B., Waltermire, D., and Scarfone, K., NIST Interagency Report 7695, *Common Platform Enumeration: Naming Specification Version 2.3,* August 2011. See http://csrc.nist.gov/publications/PubsNISTIRs.html.

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

A.2 Informative References

[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,* August 2011. See http://csrc.nist.gov/publications/PubsSPs.html#800-126.

Appendix B—Implementing CPE 2.2 Matching Functionality

This appendix explains how to implement CPE 2.2 equivalent matching capabilities using CPE 2.3 matching functions.

In [CPE22: 7.2] ("Matching Algorithm: Known Instance Based Matching") a name-comparison function `CPE_Name_Match` is defined, with the following inputs and outputs:

Inputs:

- *K* - A list of *m* CPE Names, $K = \{K_1, K_2, \ldots, K_m\}$.
- *X* - A candidate CPE Name

Output:

- True if *X* matches *K*, false otherwise.

The accompanying pseudocode is reproduced below for ease of reference:

```
function CPE_Name_Match(K, X)
  for each N in K do
    if length(N)>= length(X) then r := false.
    for i := 1 to length(X) do
      if comp(X,i) = comp(N,i) or comp(X,i) = ""
        then
          r := true.
        else
          r := false.
          break.
      end if.
    end for.
    if r = true then return true.
  end for.
  return false.
end.
```

The above function methodically compares X to each CPE name N in K, and returns TRUE when the first match is found between the source X and the target N. If no such match is found, the function returns FALSE.

The core source-to-target comparison implemented above in `CPE_Name_Match` is functionally equivalent to the `CPE_SUPERSET` function specified in Section 7.2 of this document. Using the new capabilities introduced in the current specification, one could implement `CPE_Name_Match` as follows:

Inputs:

- *K* - A list of *m* target WFNs, $K = \{K_1, K_2, \ldots, K_m\}$.
- *X* - A source WFN

Output:

- True if *X* matches *K*, false otherwise.

```
function CPE_Name_Match(K, X)
  for each N in K do
    if CPE_SUPERSET(X, N) then return true.
  end for.
  return false.
end.
```

The above pseudocode differs from the CPE 2.2 implementation in that the argument X is expected to be a WFN (rather than a CPE name encoded as a character string), and the argument K is expected to be a list of WFNs. To see that this is true, consider the following matching example, based on the example presented in [CPE2.2: 7.3]:

```
K = {wfn:[part="o",vendor="microsoft",product="windows_2000",
          update="sp3",edition="pro"],
     wfn:[part="a",vendor="microsoft",product="ie",version="5.5"]}

X = wfn:[part="o",vendor="microsoft",product="windows_2000"]
```

In this example, the result of the comparison is `true` because

```
CPE_SUPERSET(wfn:[part="o",vendor="microsoft",product="windows_2000"],
             wfn:[part="o",vendor="microsoft",product="windows_2000",
                  update="sp3",edition="pro"])
```

returns `true`. Recall that unspecified attributes in a WFN default to the logical value `ANY`, thus the value of the `update` and `edition` attributes in X are `ANY`. The result of comparing an `ANY` value in the source to a specific value (e.g., `"sp3"`) in the target is `SUPERSET` (cf. line 3 in Table 6-2), and the result of comparing an `ANY` value in the source to an `ANY` value in the target is `EQUAL` (cf. line 1 in Table 6-2). `CPE_SUPERSET` is true when all pairwise attribute comparisons yield `EQUAL` or `SUPERSET`.

Appendix C—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 – 30 June 2010

- Near final draft released to NIST for submission to review process.
- Minor editorial changes throughout the document.
- Added abstract and change log sections.
- Removed all mention of and support for the logical value UNKNOWN.
- Updated audience sections to align with the CPE Naming specification.
- Updated the name matching sections to reflect the new set relation matching results and the minimal required name matching criteria.
- Added restrictions to the wild card verbiage to allow only start and end wild card usage within a value string.
- Added source wild card to target wild card matching pseudocode.
- Broke out the name matching function to four separate functions in the pseudocode.
- Added a new section to define wild card matching criteria and methods.

Release 2 – 22 April 2011

- Reorganized the sequence of several sections and sub-sections.
- Added discussion of the set-theoretic view of WFNs that underlies the name matching model.
- Removed the INTERSECT function, and clarified the definitions of SUBSET and SUPERSET.
- Extensively revised the informative pseudocode for clarity and organization.
- Added explicit support for wild cards in the source attribute values, including informative pseudocode for the string comparison functions.
- Added Appendix B, which describes how to implement CPE 2.2-equivalent matching capabilities using CPE 2.3 functions.
- Minor editorial changes throughout the document.

Release 3 – 30 August 2011

- Final release of CPE Name Matching 2.3 specification.
- Made minor editorial changes.
- Corrected minor errors in pseudocode.