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

NIST Interagency Report 7698
(Initial Public Draft)

Common Platform Enumeration: Applicability Language Specification Version 2.3 (DRAFT)

David Waltermire
Paul Cichonski
Karen Scarfone

9

10

NIST Interagency Report 7698
(Initial Public Draft)

Common Platform Enumeration: Applicability Language Specification Version 2.3 (DRAFT)

David Waltermire
Paul Cichonski
Karen Scarfone

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

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

June 2011



U.S. Department of Commerce

Gary Locke, Secretary

National Institute of Standards and Technology

Dr. Patrick D. Gallagher, Director

Reports on Computer Systems Technology

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

National Institute of Standards and Technology Interagency Report 7698 (Initial Public Draft)
25 pages (June 2011)

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

Acknowledgments

The authors, David Waltermire and Paul Cichonski 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 of NIST, Seth Hanford of Cisco Systems, Inc., Tim Keanini of nCircle, Kent Landfield of McAfee, Inc., Brant A. Cheikes and Mary Parmelee of the MITRE Corporation, Jim Ronayne of Varen Technologies, Shane Shaffer of G2, Inc., Joseph L Wolfkiel of the US Department of Defense, and Adam Halbardier of Booz Allen Hamilton for their insights and support throughout the development of the document.

Abstract

This report defines the Common Platform Enumeration (CPE) Applicability Language version 2.3 specification. The CPE Applicability Language 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 Applicability Language data model builds on top of other CPE specifications to provide the functionality required to allow CPE users to construct complex groupings of CPE names to describe IT platforms. These groupings are referred to as applicability statements because they are used to designate which platforms particular guidance, policies, etc. apply to. This report defines the semantics of the CPE Applicability Language data model and the requirements that IT products and CPE Applicability Language documents must meet for conformance with the CPE Applicability Language version 2.3 specification.

Trademark Information

CPE is a trademark of The MITRE Corporation.

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

Table of Contents

1. INTRODUCTION 1

1.1 PURPOSE AND SCOPE 1
1.2 AUDIENCE 1
1.3 DOCUMENT STRUCTURE 2
1.4 DOCUMENT CONVENTIONS 2

2. DEFINITIONS AND ABBREVIATIONS 0

2.1 DEFINITIONS 0
2.2 ABBREVIATIONS 0

3. RELATIONSHIP TO EXISTING SPECIFICATIONS AND STANDARDS 1

3.1 OTHER CPE VERSION 2.3 SPECIFICATIONS 1
3.2 CPE VERSION 2.2 1

4. CONFORMANCE 2

4.1 PRODUCT CONFORMANCE 2
4.2 ORGANIZATION CONFORMANCE 2

5. DATA MODEL OVERVIEW 3

5.1 THE PLATFORM-SPECIFICATION ELEMENT 3
5.2 THE PLATFORM ELEMENT 3
5.3 THE LOGICAL-TEST ELEMENT 3
5.4 THE CHECK-FACT-REF ELEMENT 4

6. CONTENT DESIGN REQUIREMENTS AND RECOMMENDATIONS 5

6.1 CPE APPLICABILITY LANGUAGE DOCUMENT CONTENTS 5
6.2 THE PLATFORM ELEMENT 5
6.3 THE LOGICAL-TEST ELEMENT 6
6.4 THE FACT-REF ELEMENT 7
6.5 THE CHECK-FACT-REF ELEMENT 7

7. PROCESSING REQUIREMENTS AND RECOMMENDATIONS 8

7.1 CPE BOUND NAME CONVERSIONS 8
7.2 GRACEFUL ERROR HANDLING 8
7.3 EVALUATING ELEMENTS 8
7.3.1 Evaluating a logical-test Element 8
7.3.2 Evaluating a fact-ref Element 9
7.3.3 Evaluating a check-fact-ref Element 9

8. CPE APPLICABILITY LANGUAGE PSEUDOCODE 11

8.1 CORE FUNCTION 11
8.2 SUPPORT FUNCTIONS 12

List of Tables

Table 1. Truth Table for Logical-Test Element 9

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 consists of several modular specifications. Combinations of the specifications work together in layers to perform various functions. One of these specifications, CPE Applicability Language, defines a standardized way to describe IT platforms by forming complex logical expressions out of individual CPE names and references to checks. For example, CPE Applicability Language could combine the CPE name for an operating system (such as Microsoft Windows XP), the CPE name for an application running on that operating system (such as Microsoft Office 2007), and a reference to a check for a particular value of a certain configuration setting (such as the wireless network card being enabled in the operating system). These logical expressions are called *applicability statements*, because they are used to designate which platforms particular guidance, policies, etc. apply to. Applicability statements can be used by tools to determine whether a target system is an instance of a particular platform.

The CPE names used by the CPE Applicability Language specification are bound forms of well-formed CPE names (WFNs), which are the abstract logical constructions for CPE names [CPE23-N:5.1]. The basic building block of the CPE Applicability Language specification is referred to as the *logical test*. This is a logical conjunction (AND) or disjunction (OR) of one or more CPE names and/or references to checks. Individual logical tests can also be negated (inverted). Nested logical tests allow the user to express a platform as any logical combination of individual CPE names and/or references to checks.

Note that previous versions of CPE referred to the Applicability Language specification as simply the Language specification.

1.1 Purpose and Scope

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

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

1.2 Audience

This report is intended for two main audiences: the authors and editors of CPE Applicability Language content (applicability statements), and IT management tool developers. Readers of this report should already be familiar with CPE naming and name matching concepts and conventions, as specified in [CPE23-N] and [CPE23-M].

1.3 Document Structure

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

- Section 2 defines selected terms and abbreviations used within 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 core CPE Applicability Language data model.
- Section 6 provides CPE Applicability Language content requirements and recommendations.
- Section 7 provides CPE Applicability Language processing requirements and recommendations.
- Section 8 provides pseudocode that implements concepts defined in other sections of the specification.
- Appendix A lists normative and informative references.
- Appendix B 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 Request for Comment (RFC) 2119.[1]

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. These references use the citation convention of 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).

This specification adheres to all rules and conventions defined lower in the CPE stack of specifications. The CPE Naming Specification defines the concept of a well-formed CPE name (WFN) that is a logical representation of a CPE name [CPE23-N:5.1]. Wherever possible, this specification uses WFN representation of CPE names to limit the dependency on any particular CPE name binding.

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.

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

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

2.1 Definitions

Applicability Statement: A complex logical expression to describe an IT platform, formed out of individual CPE names and references to checks. Applicability statements are used to designate which platforms particular guidance, policies, etc. apply to.

Check Fact Reference: An expression that refers to a check (e.g., OVAL check, OCIL check).

Fact Reference: An expression that refers to a bound CPE name.

Logical Test: An expression comprised of logical operators and one or more expressions to be evaluated. The individual expressions within a logical test may be fact references, check fact references, and/or other logical tests.

Platform: In the context of the CPE Applicability Language specification only, a logical structure combining one or more bound CPE names through logical operators.

2.2 Abbreviations

CPE	Common Platform Enumeration
IR	Interagency Report
IT	Information Technology
ITL	Information Technology Laboratory
NIST	National Institute of Standards and Technology
OCIL	Open Checklist Interactive Language
OVAL	Open Vulnerability and Assessment Language
RFC	Request for Comment
URI	Uniform Resource Identifier
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 Applicability Language version 2.3 specification builds upon the CPE Naming version 2.3 [CPE23-N] and CPE Name Matching version 2.3 [CPE23-M] specifications.

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 difference between CPE Applicability Language versions 2.2 and 2.3 is the updated `namePattern` type in the version 2.3 schema, which allows for both formatted string and URI bindings of a WFN. Also, version 2.2 was called the CPE Language, not the CPE Applicability Language.

4. Conformance

Products and organizations may want to claim conformance with this specification for a variety of reasons. For example, vulnerability researchers may want to assert that they provide vulnerability applicability statements in the form of CPE Applicability Language statements. In relation to this, implementers may want to assert that their products have the ability to process applicability statements written using the CPE Applicability Language.

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

4.1 Product Conformance

All products claiming conformance with this specification MUST adhere to the following requirements:

1. If a product is reading valid CPE 2.3 Applicability Language content, then the product MUST process the content and interpret it according to the semantics of the data model defined in Section 5 and the additional requirements detailed in Section 7.

2. If the product is generating content containing CPE Language constructs, then the product MUST output the content in the form of XML instance data that validates against the CPE Applicability Language version 2.3 schema, conforms to the semantics of the CPE Applicability Language data model defined in Section 5, and conforms to the CPE name requirements defined in [CPE23-N].

3. The product MUST make an explicit claim of conformance to this specification in any documentation provided to end users.

4.2 Organization Conformance

Organizations creating or maintaining CPE 2.3 Applicability Language documents that claim conformance with this specification SHALL follow these requirements:

1. Adhere to the official CPE Applicability Language schema. Ensure that the CPE Applicability Language documents validate against the CPE Applicability Language version 2.3 schema and that the content conforms to the semantics of the CPE Applicability Language data model as defined in Section 5.

2. Adhere to the syntax, structural, and other CPE Applicability Language document development requirements defined in Section 6.

3. Adhere to the requirements for CPE names (WFNs and bound forms) as defined in [CPE23-N].

5. Data Model Overview

This section provides an overview of the data model that all CPE Applicability Language implementations MUST support. The data model does not prescribe a specific binding or implementation for itself. It merely describes the data that is required to support common CPE Applicability Language use cases. This section uses the term "element" to identify the classes within the data model, and the term "property" to identify any properties of a class. This section only defines selected aspects of the data model. The CPE Applicability Language XML schema, which is the authoritative XML binding definition, can be found at http://scap.nist.gov/schema/cpe/2.3/cpe-language_2.3.xsd. The tables below use the prefix "cpe-lang" to refer to this schema.

5.1 The platform-specification Element

The platform-specification element is the root element of a CPE Applicability Language XML document and therefore acts as a container for child platform definitions. The table below describes the platform-specification element's properties.

Element or Attribute	Type	Count	Description
platform (element)	cpe-lang:PlatformType	1-n	The description or qualifications of a particular IT platform type.

5.2 The platform Element

The table below describes the platform element's properties.

Element or Attribute	Type	Count	Description
id (attribute)	xsd:anyURI	1	A locally unique name for the platform. There is no defined format for this id; however, it must be unique within the containing CPE Applicability Language document.
title (element)	cpe-lang:TextType	0-n	A human-readable title for a platform. To support uses intended for multiple languages, the title element supports the @xml:lang attribute. At most one title element can appear for each language.
remark (element)	cpe-lang:TextType	0-n	An additional description. To support uses intended for multiple languages, the remark element supports the @xml:lang attribute. There can be multiple remark elements for a single language.
logical-test (element)	cpe-lang:LogicalTestType	1	Definition of test using logical operators (AND, OR, negate).

5.3 The logical-test Element

The table below describes the logical-test element's properties.

Element or Attribute	Type	Count	Description
operator (attribute)	cpe-lang:operatorEnumeration	1	The operator applied to the results of evaluating the fact-ref, check-fact-ref, and logical-test elements. The permitted operators are “AND” and “OR”.
negate (attribute)	xsd:boolean	1	Whether the result of applying the operator should be negated. Possible values are “TRUE” and “FALSE”. Ignored if the result of applying the operator is “ERROR”.
logical-test (element)	cpe-lang:LogicalTestType	0-n	Definition of complex logical test using AND, OR, and/or negate operators. A logical-test element evaluates to TRUE, FALSE, or ERROR.
fact-ref (element)	cpe-lang:CPEFactRefType	0-n	A reference to a bound form of a WFN; the reference always evaluates to TRUE or FALSE. The bindings contained within a fact-ref are meant to describe possible sets of products and are not meant to identify a unique product class. The fact-ref element has a mandatory attribute, name, of type cpe-lang:namePattern (defines the format for acceptable CPE names) and an optional attribute, description, of type xsd:normalizedString (contains a human-readable description of what the fact-ref checks).
check-fact-ref (element)	cpe-lang:CheckFactRefType	0-n	A reference to a check that evaluates to TRUE, FALSE, or ERROR. Examples of types of checks are OVAL and OCIL checks.

A logical-test element can be nested within another logical-test element.

Although fact-ref, check-fact-ref, and logical-test MAY be omitted from the body of a logical-test element, at least one instance of one of them MUST appear as a property of each logical-test element.

5.4 The check-fact-ref Element

The table below describes the check-fact-ref element’s properties.

Element or Attribute	Type	Count	Description
description (attribute)	xsd:normalizedString	0-1	A human-readable description of what the check-fact-ref checks.
system (attribute)	xsd:anyURI	1	The system identifier for the check system, such as *“*http://oval.mitre.org/XMLSchema/oval-definitions-5*”* or *“*http://scap.nist.gov/schema/ocil/2*”.*
href (attribute)	xsd:anyURI	1	The location of the check content, such as the OVAL or OCIL document holding the desired check.
id-ref (attribute)	xsd:token	1	The ID for the check to be evaluated, such as a specific OVAL definition ID or an OCIL questionnaire ID.

6. Content Design Requirements and Recommendations

This section defines content design requirements, including syntax and structural requirements, that an organization or product MUST follow to create and maintain a well-formed CPE Applicability Language version 2.3 document. This section also provides recommendations for CPE Applicability Language content. The requirements and recommendations presented in this section supplement the data model defined in Section 5.

6.1 CPE Applicability Language Document Contents

A CPE Applicability Language document instance SHALL hold exactly one `<cpe-lang:platform-specification>` element. The `<cpe-lang:platform-specification>` element SHALL be the root XML element of a CPE Applicability Language document.

Each CPE Applicability Language document SHALL validate against the CPE Applicability Language schema. CPE Applicability Language elements SHALL belong to the namespace "http://cpe.mitre.org/language/2.0".

6.2 The platform element

Each `<cpe-lang:platform>` element SHALL have an identifier, as expressed using the `@id` attribute, that is unique within the CPE Applicability Language document and also provides a means to uniquely reference the platform from other models besides CPE. Each `<cpe-lang:platform>` element SHALL be referenced through its identifier.

Each `<cpe-lang:platform>` element SHALL have a single `<cpe-lang:logical-test>` element defined directly within it. One or more additional `<cpe-lang:logical-test>` elements MAY be nested within any `<cpe-lang:logical-test>` element.

The following is an example of the structure of a `<cpe-lang:platform>` element:

```
<cpe-lang:platform id="789">
   <cpe-lang:title>
   Microsoft Windows XP with Internet Explorer 7.x or 8.x
   </cpe-lang:title>
   <cpe-lang:logical-test operator="AND" negate="FALSE">
      <cpe-lang:fact-ref
         name="cpe:2.3:o:microsoft:windows_xp:*:*:*:*:*:*:*:*"/>
      <cpe-lang:logical-test operator="OR" negate="FALSE">
         <cpe-lang:fact-ref
            name="cpe:2.3:a:microsoft:internet_explorer:7.%01:*:*:*:
             *:*:*:*"/>
         <cpe-lang:fact-ref
            name="cpe:2.3:a:microsoft:internet_explorer:8.%01:*:*:*:
             *:*:*:*"/>
      </cpe-lang:logical-test>
   </cpe-lang:logical-test>
</cpe-lang:platform>
```

6.3 The logical-test element

A <cpe-lang:logical-test> element MAY contain one or more <cpe-lang:logical-test> elements, MAY contain one or more <cpe-lang:fact-ref> elements, and MAY contain one or more <cpe-lang:check-fact-ref> elements. Although <cpe-lang:fact-ref>, <cpe-lang:check-fact-ref>, or <cpe-lang:logical-test> MAY be omitted from a <cpe-lang:logical-test> element, at least one instance of one of them MUST appear within the body of each <cpe-lang:logical-test> element.

A <cpe-lang:logical-test> element SHALL have an @operator attribute. This attribute SHALL be set to either "AND" or "OR". The designated operation is applied to all the values of the <cpe-lang:fact-ref>, <cpe-lang:check-fact-ref>, and <cpe-lang:logical-test> elements within the <cpe-lang:logical-test> element, resulting in a TRUE, FALSE, or ERROR value. More complex tests MAY be constructed by nesting <cpe-lang:logical-test> elements and setting their @operator and @negate attributes appropriately.

A <cpe-lang:logical-test> element SHALL have a @negate attribute. This attribute SHALL be set to either "TRUE" or "FALSE". This attribute is ignored if the value of the <cpe-lang:logical-test> element is set to "ERROR". Otherwise, if set to "TRUE", the value of the <cpe-lang:logical-test> element is inverted, and if set to "FALSE", the value of the <cpe-lang:logical-test> element is unaffected.

The following example shows the <cpe-lang:logical-test> element's syntax and structure. In this example, first the two <cpe-lang:fact-ref> elements will be evaluated. Next, an OR of their values will be done. The result will become the final value of the <cpe-lang:logical-test> element, since the negate function is not enabled.

```
<cpe-lang:logical-test operator="OR" negate="FALSE">
   <cpe-lang:fact-ref
      name="cpe:2.3:a:microsoft:internet_explorer:7.%01:*:*:*:*:*:*:*"/>
   <cpe-lang:fact-ref
      name="cpe:2.3:a:microsoft:internet_explorer:8.%01:*:*:*:*:*:*:*"/>
</cpe-lang:logical-test>
```

The truth table in Section 7.3 defines how individual results are combined by a <cpe-lang:logical-test> element into a final result.

Implementations processing CPE Applicability Language content MAY optimize their performance by skipping elements that are not necessary. For example, in the example above, if the first <cpe-lang:fact-ref> element evaluates to TRUE, then it is known that the <cpe-lang:logical-test> element will evaluate to TRUE, even though the value of the second <cpe-lang:fact-ref> element is not known. To take advantage of this efficiency, authors of CPE Applicability Language content MAY choose to plan the sequence of their <cpe-lang:fact-ref>, <cpe-lang:check-fact-ref>, and <cpe-lang:logical-test> elements, such as putting elements most likely to evaluate TRUE at the top of "OR" <cpe-lang:logical-test> elements, and putting elements most likely to evaluate FALSE or ERROR at the top of "AND" <cpe-lang:logical-test> elements.

6.4 The fact-ref element

A `<cpe-lang:fact-ref>` element SHALL have a `@name` attribute that holds a bound form of a valid WFN, as defined in [CPE23-N:6]. This SHOULD be a formatted string binding, but it MAY be a URI binding (for backward compatibility).

6.5 The check-fact-ref element

A `<cpe-lang:check-fact-ref>` element SHALL have the following attributes:

- A `@check-system` attribute that holds the system identifier for the check system. Examples of commonly used check system identifiers are "http://oval.mitre.org/XMLSchema/oval-definitions-5" for the OVAL check system and "http://scap.nist.gov/schema/ocil/2" for the OCIL check system. The `@check-system` attribute MAY refer to any check system.
- A `@check-location` attribute that holds the location of the check content, such as the OVAL or OCIL document that contains the desired check.
- A `@check-id` attribute that holds the ID for the check to be evaluated, such as an OVAL definition ID or an OCIL questionnaire ID.

When selecting a check for inclusion in CPE Applicability Language content, the author SHALL choose checks that only return results that can clearly be mapped to boolean values (TRUE and FALSE) and, optionally, error conditions (ERROR).

7. Processing Requirements and Recommendations

This section describes the processing requirements that a CPE Applicability Language implementation MUST follow to correctly process a CPE Applicability Language document.

7.1 CPE Bound Name Conversions

The CPE 2.3 Applicability Language schema defines a `cpe-lang:namePattern` type that supports two forms of bound names: formatted string bindings (defined in [CPE23-N:6.2]) and URI bindings (defined in [CPE23-N:6.1]). CPE 2.3 Applicability Language implementations MAY convert formatted string bindings into URI bindings and URI bindings into formatted string bindings. These conversions SHOULD comply with the appropriate function that implements this conversion process, either `convert_fs_to_uri` defined in [CPE23-N:7.1] or `convert_uri_to_fs` defined in [CPE23-N:7.1].

7.2 Graceful Error Handling

A CPE Language implementation SHOULD NOT suddenly terminate its execution due to an error condition. CPE Language implementations SHOULD handle all exceptional conditions gracefully by performing proper error and exception handling and producing an informative error message.

7.3 Evaluating Elements

The following subsections discuss the evaluation process for `<cpe-lang:logical-test>`, `<cpe-lang:fact-ref>`, and `<cpe-lang:check-fact-ref>` elements.

7.3.1 Evaluating a logical-test Element

The results of evaluating the individual `<cpe-lang:fact-ref>`, `<cpe-lang:check-fact-ref>`, and nested `<cpe-lang:logical-test>` elements within a `<cpe-lang:logical-test>` element are combined to produce the final value for the `<cpe-lang:logical-test>` element. Table 1 provides a truth table that defines how the results SHALL be combined. A "+" sign in the table denotes the specified value and any larger value. So, for example, "1+" means any value of 1 or greater.

Table 1. Truth Table for Logical-Test Element

Operator	Number of Individual Results (logical-test, fact-ref, check-fact-ref)			Negate	Final Result
	TRUE	FALSE	ERROR		
AND	1+	0	0	FALSE	TRUE
	1+	0	0	TRUE	FALSE
	1+	0	1+	TRUE or FALSE	ERROR
	0+	1+	0+	FALSE	FALSE
	0+	1+	0+	TRUE	TRUE
OR	1+	0+	0+	FALSE	TRUE
	1+	0+	0+	TRUE	FALSE
	0	1+	0	FALSE	FALSE
	0	1+	0	TRUE	TRUE
	0	1+	1+	TRUE or FALSE	ERROR

An implementation evaluating a `<cpe-lang:logical-test>` element MAY skip evaluating one or more of its components if their results are not necessary—for example, if a `<cpe-lang:logical-test>` element has an OR operator, then evaluation of the element's components MAY stop as soon as any component is found to be TRUE, since the final result for the element is already known. In other words, processing MAY stop as soon as there is only one row in Table 1 left as a possibility.

An implementation processing a `<cpe-lang:logical-test>` element MAY process individual components in any order; however, the implementation SHOULD process them in sequence to take advantage of sequencing choices made by content authors to help optimize evaluation performance.

7.3.2 Evaluating a fact-ref Element

For a `<cpe-lang:fact-ref>` element, the bound name in its `@name` attribute SHALL be the source matched against the target list. The target list SHALL consist of one or more bound names for valid WFNs, each of type `cpe-lang:namePattern`. An implementation MAY process the target names in any sequence. The implementation SHALL perform a CPE name match, as defined in [CPE23-M], of each target name against the source name until either a match is found or all target names have been processed. A match SHALL occur only when the source name's relation with the target name is determined to be either EQUAL or SUPERSET [CPE23-M:6.2]. A match SHALL result in a TRUE value for the `<cpe-lang:fact-ref>` element, otherwise the element SHALL be given a value of FALSE.

7.3.3 Evaluating a check-fact-ref Element

A `<cpe-lang:check-fact-ref>` element MAY use any check system that meets the basic requirements specified in Section 6.5. However, CPE Language implementations are not required to support all of these check systems. Implementations SHALL support the OVAL check system. Its use in content is indicated by the `<cpe-lang:check-fact-ref>` element's `@check-system` attribute being set to "http://oval.mitre.org/XMLSchema/oval-definitions-5". Implementations MAY support additional check systems, such as OCIL.

For all check systems that an implementation supports, the implementation MUST be capable of referencing the check using the attributes specified for the `<cpe-lang:check-fact-ref>` element and handling the result. The implementation SHALL accept results that clearly map to TRUE and FALSE.

The implementation SHALL accept error condition results, such as "error" or "not applicable", and SHALL treat them as ERROR results. Also, if an implementation does not support a check system that the `<cpe-lang:check-fact-ref>` element uses, if the specified check content is not accessible, or if any other condition occurs that prevents evaluation of the check, the implementation SHALL treat it as an ERROR result. If a result from the `<cpe-lang:check-fact-ref>` element is required to calculate a result for its parent `<cpe-lang:logical-test>` element, then evaluation of the parent element MAY cease.

8. CPE Applicability Language Pseudocode

This section specifies the required common applicability language 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. Section 8.1 defines the core function for CPE applicability language statement matching, while Section 8.2 defines supporting functions.

8.1 Core Function

The following pseudocode defines the required core CPE_App_Lang_Match function.

```
;; Begin CPE_App_Lang_Match function. Input argument E is an
;; expression in the CPE Applicability Language, represented as the
;; XML infoset for the platform element (see Section 5.2).
;; Input argument K is a set of known CPE names, represented as
;; CPE 2.3 formatted string bindings and/or URI bindings.
;; Returns the result of evaluating E (TRUE, FALSE, ERROR).
function CPE_App_Lang_Match(E, K)
  if (element(E) = "platform") then
    ;; Parse through E's elements and ignore all but logical-test.
    foreach C in children(E) do
      if (element(C) = "logical-test") then
        ;; Call the function again, but with logical-test as the
        ;; root element.
        return CPE_App_Lang_Match(C, K).
    end.
  else if (element(E) = "fact-ref") then
    ;; fact-ref's name attribute is a bound name,
    ;; so we unbind it to a WFN before passing it.
    return Fact_Ref_Eval(unbind(attribute(E, "name")), K).
  else if (element(E) = "check-fact-ref") then
    return Check_Fact_Ref_Eval(E).
  else if (element(E) = "logical-test") then
    count := 0.
    len := 0.
    answer := FALSE.
    foreach C in children(E) do
      len := len + 1.
      result := CPE_App_Lang_Match(C, K).
      if (result = TRUE) then
        count := count + 1.
      else if (result = ERROR) then
        answer := ERROR.
      end.
    end.
    if ((attribute(E, "operator") = "AND") and (count = len)) then
      answer := TRUE.
    else
      if ((attribute(E, "operator") = "OR") and (count > 0)) then
```

```
        answer := TRUE.
    if (attribute(E, "negate") = TRUE and answer != ERROR) then
      answer := ~answer.
    return answer.
  else
    return FALSE.
  endif.
end.
```

8.2 Support Functions

The following pseudocode defines the support functions needed for the core function defined in Section 8.1.

```
;; Begin Fact_Ref_Eval function. Input argument factrefname is WFN.
;; Input argument target is a list of CPE bound names.
;; Returns TRUE if factrefname is a non-proper superset (true superset
;; or equal to) any of the names in target, otherwise FALSE.
function Fact_Ref_Eval(factrefname, target)
  foreach N in target do
    ;; Need to convert each N from bound form to WFN.
    if (CPE_SUPERSET(factrefname, unbind(N)))
      return TRUE.
  end.
  return FALSE.
end.

;; Begin Check_Fact_Ref_Eval function. Input argument is the XML
;; infoset for the check_fact_ref element (see Section 5.4).
;; Returns the result (TRUE, FALSE, ERROR) of performing the specified
;; check, unless the check isn't supported, in which case it returns
;; FALSE. ERROR is a catch-all for all results other than TRUE and
;; FALSE.
;;
function Check_Fact_Ref_Eval(checkfactref)
  checksystemID := attribute(checkfactref, "check-system").
  if (checksystemID = "http://oval.mitre.org/XMLSchema/oval-
definitions-5")
    ;; Perform an OVAL check. First attribute is the URI of an OVAL
    ;; definitions file. Second attribute is an OVAL definition ID.
    return ovalcheck(attribute(checkfactref, "check-location"),
                     attribute(checkfactref, "check-id")).
  if (checksystemID = "http://scap.nist.gov/schema/ocil/2")
    ;; Perform an OCIL check. First attribute is the URI of an OCIL
    ;; questionnaire file. Second attribute is OCIL questionnaire ID.
    return ocilcheck(attribute(checkfactref, "check-location"),
                     attribute(checkfactref, "check-id")).
  ;; can add additional check systems here, with each returning a
  ;; TRUE, FALSE, or ERROR value
  return FALSE.
```

```
end.

;; Unbinds a bound form to a WFN.
function unbind(boundname)
  if fs(boundname)
    return(unbind_fs(boundname)).
  else
    return(unbind_URI(boundname)).
end.

;; Pseudocode for unbind_fs function is at [CPE23-N:6.2.3.2].
function unbind_fs(fs)
  ;; Unbinds a formatted string fs to a WFN.
  ;; Input is formatted string, output is WFN.
end.

;; Pseudocode for unbind_URI function is at [CPE23-N:6.1.3.2].
function unbind_URI(uri)
  ;; Unbinds a URI binding uri to a WFN.
  ;; Input is URI binding, output is WFN.
end.

;; Pseudocode for CPE_SUPERSET is at [CPE23-M:7.2].
function CPE_SUPERSET(source,target)
  ;; Returns TRUE if the set relation between source and target is
  ;; SUPERSET, otherwise FALSE. Input arguments are WFNs.
end.

;; Makes a structure for parsing a CPE Applicability Language
;; expression.
function children(langexpr)
  ;; Holds the XML elements of the expression so that they can be
  ;; parsed one at a time.
  ;; Input is CPE Applicability Language expression, represented
  ;; as the XML infoset for the platform element.
  ;; Output is the structure holding the expression elements.
end.

;; Checks the element type (platform, logical-test, etc.) of a
;; CPE Applicability Language expression element.
function element(langexprelement)
  ;; Input is one element of a CPE Applicability Language
  ;; expression, from the structure created by the children function.
  ;; Output is the type of that element, represented as a string.
end.

;; Returns the value of a particular attribute within a
;; CPE Applicability Language expression.
function attribute(langexpr, attribtype)
  ;; Input is a CPE Applicability Language expression and an
  ;; attribute type (operator, negate, etc.)
  ;; Output is a string representing the value of that attribute
```

```
625     ;; (for example, the operator type could have a value of "AND"
626     ;; or "OR".
627   end.
628
```

Appendix A—References

The following documents are indispensible 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-D] Cichonski, P., Waltermire, D., and Scarfone, K., NIST Interagency Report 7697, *Common Platform Enumeration: Dictionary Specification Version 2.3 (Second Public Draft)*, June 2011. See http://csrc.nist.gov/publications/PubsNISTIRs.html .

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

[CPE23-N] Cheikes, B., Waltermire, D., and Scarfone, K., NIST Interagency Report 7695, *Common Platform Enumeration: Naming Specification Version 2.3 (Draft),* April 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/nistpubs/800-117/sp800-117.pdf.

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

Appendix B—Change Log

2.3 DRAFT 1 – 2 June 2011

- Initial draft specification released for public comment.