Common Platform Enumeration (CPE) – Specification

Andrew Buttner, The MITRE Corporation
Neal Ziring, National Security Agency

Version 2.2 - 11 March 2009

Approved for Public Release; Distribution Unlimited.
07-0030

1. Introduction

Following security best practices is essential to maintaining the security of IT systems. To this end, several specification languages exist for describing vulnerabilities, testing system state, and expressing security checklists. But descriptions of vulnerabilities and configuration best practices have greater utility when all participants share common names for the entities described. Further, use of consistent and meaningful names can speed application development, foster interoperability, improve correlation of test results, and ease gathering of metrics.

All vulnerability and configuration information items have an important distinction that affects their use: they apply only to a particular range of IT systems, platforms, or applications. This is so obvious that IT managers and security administrators sometimes forget about how critical it can be. When a new vulnerability is announced, the first question most practitioners will ask is: "which systems are vulnerable?" In prose vulnerability descriptions, informal or colloquial names for IT platforms are adequate. Experienced system administrators and security analysts can understand and use ad hoc names.

There is a strong trend toward automation in security practice. Automated systems cannot work with informal or ad hoc names. To foster effective automation, the community needs a more formal, consistent, and uniform naming scheme that allows tools (as well as human analysts and authors) to clearly identify the IT platforms to which a vulnerability or element of guidance applies.

Today, a popular and widespread naming scheme exists for vulnerabilities; the Common Vulnerabilities and Exposures (CVE) naming scheme is widely used for identifying and describing IT platform vulnerabilities. A somewhat similar scheme also exists for IT platform configuration statements: the Common Configuration Enumeration (CCE).

This specification describes a structured naming scheme for IT platforms (hardware, operating systems, and applications): the Common Platform Enumeration (CPE). It is based on the generic syntax for Uniform Resource Identifiers [2]. The CPE Specification includes the naming syntax and conventions for constructing CPE Names from product information, a dictionary (and associated XML Schema) that holds a collection of all known CPE Names as well as a binding of descriptive and diagnostic information, a language for creating complex platform descriptions, and a matching algorithm. For the up-to-date CPE Dictionary, and for the complete list of abbreviations and formatted names to be used, please visit the CPE web site at:

`http://cpe.mitre.org/`

Using a clear and uniform naming specification, community members will be able to generate names for new IT platforms in a consistent and predictable way.

2. Use Cases

The following list of use cases is meant to help define the purpose of CPE.

2.1. Common Names

One tool marks a statement/result as applicable to a certain type of platform(s). A second tool imports the statement/result and takes a specific action based on the type of platform(s). The second tool must be able to understand what the first tool is 'saying' regarding the platform type so it can know which action to perform. In other words, the second tool must use the same term to describe the specific platform as the first tool.

Note that the above example is very "tool" centric. This use case also applies to web services (SOA), data dictionaries, and other data related uses.

2.2. Matching

A configuration guide might be applicable for all versions of a certain Operating System (OS) platform and be assigned a CPE Name that represents this. A tool might query a system and determine a more specific CPE Name that includes an exact version, or maybe determine a complex platform description utilizing the CPE Language. When the guide is evaluated, the tool must be able to determine that the CPE Name (or CPE Language statement) assigned to the system is a member of the more general CPE Name assigned to the guide.

2.3. Reporting

When used in reports about a specific system or a group of systems, a CPE Name allows correlation of results with additional sources. For example, a report may be generated about existing systems identified on a given network. If a CPE Name is provided for each system, that identifier can be used to research a 3rd party site about possible vulnerabilities that are applicable for that type of platform.

3. Terms

The following terms are used throughout this specification:

CPE Name: A unique collection of components given to a specific platform type.

Platform: An IT system that is made up of hardware, applications, an operating system, and other possible parts.

Platform Part: An individual section of high-level platform information. Many parts have been identified: hardware, operating system, application.

Component: A specific facet of platform information. For example, a vendor name or a version number.

CPE Dictionary: A collection of official CPE Names along with any necessary supporting information (title, references, automated check, etc.)

CPE Language: A method for combining multiple CPE Names to describe a complex platform type.

Matching: The process of determining if a given CPE Name or CPE Language statement specifies a platform that is defined by a set of known CPE Names.

4. Requirements

For naming IT platforms subject to vulnerability and configuration guidance, patching and remediation, asset management, and other security related tasks, there are many distinct parts of a platform that need to be addressed.

Hardware– the physical platform supporting the IT system. The type and model of hardware can be relevant for some guidance and vulnerabilities.

Operating System – the operating system controls and manages the IT hardware and supports applications. The operating system type, version, edition, and upgrade status are almost always relevant for vulnerability descriptions and guidance.

Application Environment – software systems, servers, and packages installed on the system are often relevant for vulnerability assessment and guidance. The diversity of applications that may be installed on a modern IT platform is very great, but typically a specific piece of guidance or a specific vulnerability description depends on only one or two applications.

- A CPE Name MUST be able to express each type of platform part described above.

Security guidance information and vulnerability information apply to a wide range of product and system categories. For example, some vulnerabilities affect an entire product line, while others affect only a particular product release or version. Some guidance applies to entire product family or system type, but other guidance can apply only to a particular version of an application running on a particular OS release.

- CPE MUST be able to express platform information across a wide range of specificity.

Note that the use of broader CPE Names requires less verbosity but also is not very "future proof", meaning that new versions of a platform may be released that have changed in such a way that the original guidance associated with the broad CPE Name does not apply. Again, this can be very dangerous and used with caution.

Also note that this wide range of specificity is not meant to enable the transferring of information about a system. In other words, CPE is not intended to be used to used to share information characterizing a specific system, it is designed to enumerate different platform types.

- The CPE Specification SHALL focus on enumerating platform types.

There are many real world cases where the distinction between languages is significant and necessary when defining a target platform.

- A CPE Name MUST be able to include the language a particular platform supports.

Along with expressive names, support for automated checking of systems against the named platform type should be defined in the current specification. The CPE Dictionary can include a link with each CPE Name to a check written in the Open Vulnerability and Assessment Language (OVAL [3]); this check can be used, with respect to an IT system, to determine whether the system is an instance of the named platform.

- The CPE Specification MUST define some means to specify concrete platform identification tests.

A general requirement for the naming structure is that the set of platforms identified by a long name should be a subset of the set of platforms identified by a shorter initial portion of that same name, thus allowing matching to take place. This is called the "prefix property". For example, redhat:enterprise_linux:4 would be a subset of redhat:enterprise_linux.

- Each CPE Name MUST exhibit the prefix property.

Additionally, CPE Names should be specified such that there is a single naming convention producing a single abbreviation for relevant terms, instead of allowing multiple different names that all mean the same thing. The enforcement of a single name will be accomplished through the specification and the official CPE Dictionary. Where the specification does not define specific structure, (for example, information beyond the vendor/product/version components) one should refer to the CPE Dictionary to make sure a similar name does not already exist.

- The CPE Specification MUST enforce the creation of unique terms for a given name.

Finally, many IT vulnerabilities and guidance items depend on configuration, but that is not part of what CPE provides. For example, the desired value of a particular setting is not something to be described using CPE. Another example of something not targeted by CPE is whether a particular service is enabled or not. CPE is meant to provide a structured name for a given platform type: installed hardware, operating system, and applications.

- CPE Names SHALL NOT be used to express aspects of system configuration.

One of the goals of the CPE Specification is to outline a way for unique names to be created in a defined way. For example, if two people were to try to create a new CPE Name for the same application, the hope is that by following the specification, that they will come up with the same name. It is realized though that this is unrealistic for a requirement. Instead, the CPE Specification is used more as a guide to help in the creation of new CPE Names.

5. CPE Names

This section explains the syntax for the representation of CPE Names, and provides several illustrative examples. Each CPE Name defines a set of platform types. Another way of thinking about a CPE Name is that it identifies a bucket of related platform types. How each bucket relates to an external entity is beyond the scope of CPE, and is something that the data producer should define. All CPE is trying to do is define these buckets and present a common name for each.

For example, consider two security configuration guides for Windows XP each written by a different producer. One producer may choose to tag the guide with the union of all CPEs for all known versions of XP for which the guide applies, with the expectation of updating the CPE tags as appropriate if/when new versions of XP become available. But the other producer may choose to tag the guide with "Windows XP", despite the fact that the guide does not apply to some current or future versions of Windows XP. Both approaches are consistent with the CPE specification, but what can be assumed by users about the relationship between the guides and certain platform types is very different. The user must obtain this information from the guide author.

This section of the specification helps explain how to create a CPE Name for a given bucket. When questions arise about terms to be used, the CPE Dictionary should be consulted for any existing related names whose terms could be copied. If the dictionary does not provide any help, then the CPE Community should help decide what is the best route forward for a particular name. For example, if there is a question about what to use as the Product Component, polling the CPE Community should be considered. In the end, all that really matters is that the CPE Name is unique.

5.1. Basic Structure

A CPE Name is a percent-encoded URI [2] with each name starting with the prefix (the URI scheme name) “cpe:”. Note that the scheme "cpe:" is not registered as an official URI scheme with IANA.[1]

Each platform can be broken down into many distinct parts. A CPE Name specifies a single part and is used to identify any platform that matches the description of that part. The distinct parts are:

- *Hardware Part* - This part identifies hardware aspects of the IT platform and denotes things like a chassis, module, card, or other physical aspect of a platform. This can include virtualized or shared hardware.

[1] Registration is something that might be considered in the future, but currently is not seen as something worth the effort to accomplish. It is unknown what the benefits of registration would be.

- *Operating System Part* - This part identifies operating system aspects of the IT platform by describing an operating system running on the platform. Note that multi-processor devices and distributed systems may incorporate multiple operating systems.

- *Application Part* - This part identifies relevant applications, services, and packages installed on the IT platform. If a system includes multiple applications, then multiple CPE Names may apply to it.

The specific part being enumerated is distinguished by a single letter code at the beginning of the name. The next section defines the syntax of individual components fully.

Note that the distinction between OS and Application is currently a bit of a grey area. The argument could be made that the OS is just another application. Also, the difference between editions of an OS is often just the packages (applications) that are bundled with it. In this case, should there be different CPE Names for the different editions of the OS? Or should it just be a single OS name and a collection of Application names? This is an area that will hopefully become better defined as the community uses the specification and more is learned about the issue. For now, a CPE Name will be available for each edition as this is more natural for the community and aligns with previous naming conventions. The individual packages can also be identified by specific application names.

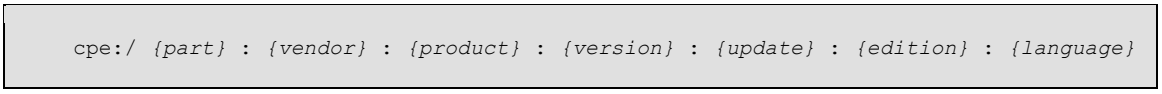
5.2. Component Syntax

The syntax of CPE Names is tightly controlled to foster creation and maintenance of unique names. One of the requirements of CPE is that there be only one way to state a specific platform name and to accomplish this, the exact syntax and abbreviations used must be strictly enforced. This section tries to explain the logic that is followed when creating a new CPE Name. Careful application of this logic will help keep new CPE Names following a consistent naming convention. Unfortunately, due to the sheer quantity of possible CPE Names, the entire list of CPE Names can not be included in this specification. Please visit the CPE web site for an up to date dictionary of official CPE Names.

CPE names are case-insensitive. The components “WebLogic” and “weblogic” are equivalent. To reduce potential for confusion, all CPE Names should be written in lowercase.

Each CPE Name consists of one or more components, separated by colons. The first component identifies the platform part being specified, the second component identifies a vendor or supplier of the element, the third component identifies the product name, the forth component states a version of the product, the fifth component is used for update or service pack information, the sixth component identifies an edition, and the seventh component is used to specify internationalization information. Figure 1 shows the basic syntax of a CPE Name.

Figure 1 – CPE Name Structure



Empty components are legal; they designate a part of the platform name for which any aspect of the platform is valid. For example, the CPE Name below designates all Professional editions of Microsoft Windows XP, regardless of service pack level.

```
cpe:/o:microsoft:windows_xp:::pro
```

Note that the above example does not identify a platform with Windows XP and no Service Pack (i.e. the initial release of XP), but rather it identifies a platform with Windows XP regardless of which service pack (or no service pack) has been installed. This will be discussed more in the Update Component section.

Also note that in an effort to provide consistency and keep the specification simple, no distinction is made between components that can and can not be left empty. At this time it is unknown if there is a meaningful name that leaves that Part component empty, but it has been decided that the consistency is more important in this case.

It is often necessary to use a CPE Name when identifying a specific release of a given platform. If attempting to create a CPE Name for this, and a specific component is not applicable to the given platform, then the term '-' should be used. Note that use of the '-' term is different than leaving the component blank, even though in practice both options might identify with the same set of platform types. For example, an application may not have different editions. A CPE Name for such an application may use the '-' term for the edition component.

```
cpe:/a:acme:product:1.0:update2:-:en-us
```

The platform type that the CPE Name above identifies may initially be the same as the platform type identified by the following CPE Name that uses a blank edition component:

```
cpe:/a:acme:product:1.0:update2::en-us
```

Remember that the blank component is used to identify the platform type regardless of the blank component. So in this case, it would identify every edition of the "Acme Product 1.0 Update 2 English". The problem is that in the future, the vendor may in fact decide to release a new edition of the product. For example:

```
cpe:/a:acme:product:1.0:update2:pro:en-us
```

Looking back at the first two examples, the CPE Name that used the '-' term would not match this new release, while the CPE Name that used the blank component would match.

The '-' term might also be used to describing an initial release if a platform does identify with a given component but no term is commonly used to describe it. An example of this might be the initial release of an application before any updates have been released. The very first release of an application might be known as "Acme Product 1.0". A few months later an update is released by the vendor and the once applied, the platform is known as "Acme Product 1.0 Update 1". In this case the two CPE Names would be:

```
cpe:/a:acme:product:1.0:-
cpe:/a:acme:product:1.0:update1
```


Again note that the CPE Name cpe:/a:acme:product:1.0: with a blank update component would match both of the names above.

It is important to note that when a CPE Name is being created for a specific platform type (i.e. a specific release of an application) then the hyphen should be used at all times (when a vendor term is not available) instead of a blank component. The blank component is to be used only when a more general platform type is being enumerated. (i.e. any version of an application)

Part Component

The first component in a CPE Name is a single letter code that designates the particular platform part that is being identified. The following codes are defined for CPE 2.0.

```
h  =  hardware part
o  =  operating system part
a  =  application part
```

Additional character codes may be added as necessary in a future version of this specification. For example, maybe 'd' for driver, 'l' for library, 'r' for runtime environment, or v for virtualization.

Vendor Component

The second component of a CPE Name is the supplier or vendor of the platform part. The vendor component of a name might be a source of ambiguity because there are many ways to express the names of companies and other organizations. For CPE, the name used for a supplier should be the highest organization-specific label of the organization's DNS name. Even if the domain name is different than the company name, it is still recommended to use the domain name for the CPE Name. The table below shows some representative examples.

Organization Full Name	DNS Domain	CPE component
Cisco Systems, Inc.	cisco.com	cisco
The Mozilla Foundation	mozilla.org	mozilla
University of Oxford	oxford.ac.uk	oxford

Note that abbreviations should not be used in the vendor component and instead the DNS name should be used as found.

Many big organizations register multiple names (e.g. "hewlett-packard.com" and "hp.com") but most of the time, the company uses one of them as the official name in their advertisements and documentation. That is the name CPE should use. If it is still ambiguous after that, then CPE will need to get guidance from the vendor or community. Using the full corporate name would create much longer names, as well as potential character encoding issues.

If two different vendors share the same organization-specific label but with a different DNS suffix then the full DNS name should be used. For example, if cpe:/a:acme already exists in the CPE Dictionary and this refers to the vendor site www.acme.com, then a new name for the vendor www.acme.org should have a CPE Name of cpe:/a:acme.org.

In some cases, especially with open source software, a vendor may not have a qualified DNS name. For these situations, the term used in the vendor component should be formed using the most widely known form of the vendor name, replacing spaces with underscores. For example, the vendor Best Software may not have a qualified DNS name, so a CPE Name for their application ABC123 would be as follows:

```
cpe:/a:best_software:abc123
```

For applications that do not have a vendor or organization associated with them, this component should use a developer's name. For example, there are a number of shareware tools that have been developed by an individual and posted to the web. They don't have a vendor, just a name of a developer. Note that multi-word names should use underscores instead of spaces.

```
cpe:/a:jon_smith:tool_name:1.2.3
```

For situations when a vendor name changes due to marketing or acquisition reasons, the original CPE Name(s) with the old vendor name will remain unchanged. A new CPE Name(s) with the new vendor name will be created for new products released under the new name. Although this process breaks the ability for matching to occur across the vendor name transition, the simplicity of the process is felt to be a worthwhile benefit.

Product Component

The third component of a CPE Name is the product name of the platform part. To determine the string to use for the product component, one should try to find the most common and recognizable name for the product. Possible things to consider are marketing materials, API return values, product documentation, etc. Unfortunately there is no clear cut definitive way of doing this, so help from the vendor and/or community will often be needed.

Multi-word product names and designations should be spelled out in full, replacing spaces with underscores. The example below shows how this would look for the Zone Labs ZoneAlarm Internet Security Suite version 7.0.

```
cpe:/a:zonelabs:zonealarm_internet_security_suite:7.0
```

Multi-word product names may be shortened when doing so would not make the CPE Name ambiguous and when the vendor has designated a particular "official" abbreviation in product descriptions. This helps keep the name more reasonable in length. For example, "Internet Explorer" should be abbreviated as "ie", and "Java Runtime Environment" should be abbreviated as "jre". A list of community product name abbreviations will be maintained at the CPE web site.

Product Name	CPE Abbreviation

Product Name	CPE Abbreviation
Internet Explorer	**ie**
Java Runtime Environment	**jre**

As with the vendor component, if a product has a name change, existing CPE Names should not be modified. Rather, new names that are created with a new version of the product should use the new product name.

Version Component

The forth component of a CPE Name is the version of the platform part. The version should be represented in the same format as seen within the product. For example, use periods, dashes, etc. as the delimiter in the same way as the product.

The following example denotes Adobe Reader version 8.1

```
cpe:/a:adobe:reader:8.1
```

Note that there is currently no way to separate major and minor versions into different components. If a CPE Name is needed for all Adobe Reader applications that have a major version of 8, then a separate CPE should be created as follows:

```
cpe:/a:adobe:reader:8
```

Update Component

The fifth component of a CPE Name is used for update or service pack information. Sometimes this is referred to as point releases or minor versions. The technical difference between version and update will be different for certain vendors and products. The use of existing CPE Names should be used to determine what is best for new names.

The following example denotes Red Hat Enterprise Linux 4.0 Update 4.

```
cpe:/o:redhat:enterprise_linux:4:update4
```

Many times, products are initially released without a defined update. For example there is not an "update 0" for Enterprise Linux and there is not a "Service Pack 0" for Microsoft Windows 2000. If it is desired to specifically identify this initial release, then the vendor's term for initial release should be used in the update component. For example, Red Hat uses the term "ga" for General Availability, and the CPE Name for the initial release of Enterprise Linux 4 would be:

```
cpe:/o:redhat:enterprise_linux:4:ga
```

If there is no vendor term for an initial release, or if there is no commonly used term, then the '-' character should be used when creating a CPE Name for that initial release.

Edition Component

The sixth component of a CPE Name is the edition of the platform part. Abbreviations should be used where appropriate. Please see the section 5.3 for a list of valid abbreviations.

The following example denotes all versions of Microsoft Windows 2000 Service Pack 4 Professional Edition.

```
cpe:/o:microsoft:windows_2000::sp4:pro
```

The Edition Component is also used to define specific target hardware and software architectures that need to be named. In a way, we are treating these as different editions of the platform. For a hardware example, an application may be developed for both the i386 and x64 architectures, and each edition needs a different name. For a target software implementation an application may be developed to run on Windows, Linux, or Macintosh OS X.

Note that the idea of a target hardware or software architecture is different than the actual hardware or software of the platform. The notion described above is used to distinguish and identify different editions of the given platform part.

Target hardware and software environment names will be appended to the end of any other names in the Edition component. Where both target hardware and software architectures are named, the software name will be listed first.

Language Component

The seventh and final component of a CPE Name is the language associated with the specific platform. This component should be represented by a valid language tag as defined by IETF RFC 4646 entitled *Tags for Identifying Languages*. [6] Note that while any valid language tag is acceptable, CPE Names should generally only use tags containing language and region codes.

The following example is the CPE Name associated with a platform that has the traditional Chinese version of Mozilla Firefox version 2.0.0.6 for the Mac OSX operating system. Note that since this CPE Name pertains to all updates of this version the update component has been left blank.

```
cpe:/a:mozilla:firefox:2.0.0.6::osx:zh-tw
```

For a complete BNF grammar that defines what is acceptable in a CPE Name, please see Appendix A.

5.3. Abbreviations

Some words and phrases appear frequently in IT product designations. Including these common words in CPE Names would be wasteful, and contrary to the objective of making CPE Names compact and easy to use. To help shorten some of the longer CPE Names, abbreviations are used where appropriate. Note that each phrase must be associated with only one abbreviation.

The table below lists abbreviations for common terms and multi-word constructions that should be applied in selecting CPE Name components. For an up-to-date list of abbreviations, please visit the CPE web site at http://cpe.mitre.org/.

Full Designation	CPE Abbreviation
advanced	**adv**
professional	**pro**
server	**srv**
standard	**std**
edition	**ed**
version 3.4	**3.4**
patch level 3	**pl3**
release 3	**r3**
release candidate 2	**rc2**
service pack 4	**sp4**
support pack 2	**sup2**
service release 2	**sr2**
security rollup	**sru**
general availability	**ga**

5.4. Percent Encoding

A CPE Name is represented by a URI that is in its post-encoded form. In other words, the standard URI percent-encoding mechanism must be used to represent a number of reserved characters. A percent-encoded octet is encoded as a character triplet, consisting of the percent character "%" followed by the two hexadecimal digits representing that octet's numeric value. Please note that if the "%" character is to be used outside of this encoding, then it would itself have to be encoded.

The following is a list of characters that must be percent encoded if they are used within a component:

- colon (**:**) – %3A – used to separate the components in a name
- slash (/) – %2F – URI reserved character, gen-delims

- question mark (**?**) – %3F – URI reserved character, gen-delims
- pound sign (**#**) – %23 – URI reserved character, gen-delims
- open bracket (**[**) – %5B – URI reserved character, gen-delims
- close bracket (**]**) – %5D – URI reserved character, gen-delims
- at sign (**@**) – %40 – URI reserved character, gen-delims
- exclamation point (**!**) – %21 – URI reserved character, sub-delims
- dollar sign (**$**) – %24 – URI reserved character, sub-delims
- ampersand (**&**) – %26 – URI reserved character, sub-delims
- apostrophe (') – %27 – URI reserved character, sub-delims
- open parenthesis (**(**) – %28 – URI reserved character, sub-delims
- close parenthesis (**)**) – %29 – URI reserved character, sub-delims
- asterisk (*****) – %2A – URI reserved character, sub-delims
- plus sign (**+**) – %2B – URI reserved character, sub-delims
- comma (**,**) – %2C – URI reserved character, sub-delims
- semi-colon (**;**) – %3B – URI reserved character, sub-delims
- equal sign (**=**) – %3D – URI reserved character, sub-delims
- percent-sign (**%**) – %25 – used for character encoding in URIs.
- angle bracket (<) – %3C – used in XML
- angle bracket (>) – %3E – used in XML
- double quote (") – %22 – used in XML

The percent-encoding mechanism is a frequent source of variance among otherwise identical URIs. Some URI producers percent-encode octets that do not require percent-encoding, resulting in URIs that are equivalent to their non-encoded counterparts. These URIs can be normalized by decoding any percent-encoded octet that corresponds to an unreserved character (unreserved = ALPHA / DIGIT / "-" / "." / "_" / "~").

When matching is performed, each component should be decoded when used during comparisons. Note that the components of CPE Name must be parsed and separated before the percent-encoded octets within those components can be safely decoded, as otherwise the data may be mistaken for component delimiters.

5.5. Examples

This example name below represents a typical CPE Name that refers to the Microsoft Windows 2000 operating system, all editions and update levels.

```
cpe:/o:microsoft:windows_2000
```

This example CPE Name below identifies Microsoft's Windows XP operating system, Professional Edition, update level "Service Pack 2".

```
cpe:/o:microsoft:windows_xp::sp2:pro
```

This example name below refers to Red Hat Enterprise Linux 3 Advanced Server.

```
cpe:/o:redhat:enterprise_linux:3::as
```


This example specifies an application, specifically the Apache Foundation HTTP server version 2.0.52.

```
cpe:/a:apache:httpd:2.0.52
```

This example name below refers to a particular web browser, regardless of any hardware or particular OS on which it is running.

```
cpe:/a:microsoft:ie:6.0
```

This example name below refers to a Cisco model 3825 integrated services router.

```
cpe:/h:cisco:router:3825
```

The example name below identifies a particular laptop computer hardware platform. The vendor is Dell Computer, the product line name is "Inspiron", and the version (or model in this example) number is 8500. Note that the platform enumerated below might be made up of a number of different peripherals, each assigned their own CPE Name if needed.

```
cpe:/h:dell:inspiron:8500
```

CPE Names may also be used to identify virtual hardware, in the context where the virtual hardware platform serves the same role as a similar real hardware platform. The CPE Name below shows an example for a virtual hardware platform.

```
cpe:/h:emc:vmware_esx:2.5
```

It is quite possible that a vendor, perhaps for marketing purposes, may use different names for identical hardware and software platforms. This could occur for regional versions with different languages, or different vertical markets each with a need for the same tool. In these cases, a different CPE Name will be used for each of the different vendor-assigned names. CPE in general tries to mimic the product naming structures used by the vendors, while imposing uniform structure and matching semantics.

6. CPE Language

An individual CPE Name addresses a single part of an actual system. To identify more complex platform types, there needs to be a way to combine different CPE Names using logical operators. For example, there may be a need to identify a platform with a particular operating system AND a certain application. The CPE Language exists to satisfy this need, enabling the CPE Name for the operating system to be combined with the CPE Name for the application.

The CPE Language can be thought of as an extension of the CPE Name syntax above. The syntax for CPE Names has been kept simplistic and restrictive, thus allowing easy adoption and quick acceptance within the community. The CPE Language on the other hand is meant to absorb the complexity that was left out of CPE Names. This will result in more powerful platform identification. Changes as the specification evolves will be limited to the language, allowing users to rely on the stability of CPE Names.

The CPE Language is essentially a Boolean expression that (1) allows users to qualify applicable platforms, and (2) allows testing tools to determine whether any target system is an instance of that platform.

6.1. Structure

The basic building block of the CPE Language is referred to as the Logical Test. This is a logical conjunction (AND) or disjunction (OR) of one or more CPE Names. Individual logical tests can also be negated (inverted). Nested logical tests allow the user to express the platform as any logical combination of individual CPE Names.

6.2. XML Representation

This section defines a concrete representation of the CPE Language in XML using both core XML syntax and XML Namespaces. CPE Language elements belong to the namespace "http://cpe.mitre.org/language/2.0". The recommended namespace prefix is "cpe".

The example below gives an idea of how the XML representation of the CPE Language is used to identify a complex platform.

Figure 2 – example XML representation

```
<?xml version="1.0"  encoding="UTF-8"?>

<cpe:platform-specification
xmlns:cpe="http://cpe.mitre.org/language/2.0">

    <cpe:platform id="123">
        <cpe:title>Microsoft Windows XP with Adobe Reader</cpe:title>
        <cpe:logical-test operator="AND" negate="FALSE">
            <cpe:fact-ref name="cpe:/o:microsoft:windows_xp" />
            <cpe:fact-ref name="cpe:/a:adobe:reader" />
        </cpe:logical-test>
```



```
    </cpe:platform>

    <cpe:platform id="456">
        <cpe:title>Sun Solaris 5.8 or 5.9 or 5.10</cpe:title>
        <cpe:logical-test operator="OR" negate="FALSE">
            <cpe:fact-ref name="cpe:/o:sun:solaris:5.8" />
            <cpe:fact-ref name="cpe:/o:sun:solaris:5.9" />
            <cpe:fact-ref name="cpe:/o:sun:solaris:5.10" />
        </cpe:logical-test>
    </cpe:platform>

    <cpe:platform id="789">
        <cpe:title>Microsoft Windows XP with Office 2003 or
2007</cpe:title>
        <cpe:logical-test operator="AND" negate="FALSE">
            <cpe:fact-ref name="cpe:/o:microsoft:windows_xp" />
            <cpe:logical-test operator="OR" negate="FALSE">
                <cpe:fact-ref name="cpe:/a:microsoft:office:2003" />
                <cpe:fact-ref name="cpe:/a:microsoft:office:2007" />
            </cpe:logical-test>
        </cpe:logical-test>
    </cpe:platform>

</cpe:platform-specification>
```

The rest of this section describes the individual elements that may appear in the XML representation of the CPE Language. For full details, see the schema presented in Appendix B.

<platform-specification>

This element is the root element of a CPE Language XML documents and therefore acts as a container for child platform definitions.

Content:	elements
Cardinality:	1
Parent Elements:	*none*
Attributes:	*none*
Child Elements:	<platform>

<platform>

The platform element represents the description or qualifications of a particular IT platform type. The platform is defined by one or more logical-test child elements. The id attribute holds a locally unique name for the platform. There is no defined format for this id, it just has to be unique to the containing CPE Language document.

Content:	elements
Cardinality:	1-n
Parent Elements:	<platform-specification>
Attributes:	id
Child Elements:	<title>, <remark>, <logical-test>

<title>

The optional title element may appear as a child of a platform element, to provide a human-readable title for it. To support uses intended for multiple languages, this element supports the 'xml:lang' attribute. At most one title element can appear for each language.

Content:	string
Cardinality:	0-n
Parent Elements:	<platform>
Attributes:	xml:lang
Child Elements:	*none*

<remark>

The optional remark element may appear as a child of a platform element. It provides some additional description. Zero or more remark elements may appear. To support uses intended for multiple languages, this element supports the 'xml:lang' attribute. There can be multiple remarks for a single language.

Content:	string
Cardinality:	0-n
Parent Elements:	<platform>
Attributes:	xml:lang
Child Elements:	*none*

<logical-test>

The logical-test element appears as a child of a platform element, and may also be nested to create more complex logical tests. The content consists of one or more elements: fact-ref, and logical-test children are permitted. The operator to be applied, and optional negation of the test, are given as attributes.

Content:	elements
Cardinality:	0-n
Parent Elements:	<platform>, <logical-test>
Attributes:	operator, negate
Child Elements:	<fact-ref>, <logical-test>

NOTE - The following operators are permitted: "AND", "OR"

<fact-ref>

The fact-ref element appears as a child of a logical-test element. It is simply a reference to a CPE Name that always evaluates to a Boolean result.

Content:	*none*
Cardinality:	0-n
Parent Elements:	<logical-test>
Attributes:	name
Child Elements:	*none*

7. Matching

Matching is the process of determining if a given CPE Name or CPE Language statement specifies a platform that is defined by a set of known CPE Names.

For example, when a checking or analysis tool tests a real IT target system, it must be able to decide whether any given CPE Name corresponds to that system, and vice-versa. If a CPE Name has an associated OVAL Definition, and the definition evaluates to true, then the system is said to be an *instance* of that name. But not every CPE Name will have an OVAL Definition associated with it. This is especially true if a CPE Name has been created by an individual author or application to identify a new or unusual platform, or if the CPE Language has been used to describe a complex platform. For these cases, there is a defined process for how to match a CPE Name or a CPE Language description against an actual system based on other known CPE Names (ones that have been matched via an OVAL Definition). This is called *matching*.

Matching helps define the relationship between different CPE Names (or language statements) and follows the hierarchical relationship built into the naming format. It is important to note that it will often be the case that different relationships between CPE Names and external data elements will also be asserted. The "tagging" of data elements with CPE Names does not imply the type of external relationship being used. It is important that the data producer defines the type of relationship it has with CPE. It is also important to note that matching does not carry over to this relationship with external enties and that one has to be careful about assuming the hierarchical relationship between CPE Names would also apply to the external relationship. It is not always the case.

By way of an example, consider a vulnerability that has been "tagged" with the CPE Name cpe:/o:microsoft:windows_xp. One can not assume that this vulnerability applies to an XP Service Pack 2 system as the relationship may have been defined as follows: "the vulnerability may or may not apply to a platform matching the CPE Name, but definitely does not apply to those that don't match". Even though cpe:/o:microsoft:windows_xp::sp2 matches the original CPE Name used by the tag, the same relationship outlined above does not hold. Think about Service Pack 1 ... it is possible according to the original relationship that the vulnerability might affect this platform, but trying to follow the same relationship with the CPE Name related to service pack 2 would say that sp1 does not apply, a contradiction would exist.

Remember that the defined relationship between a CPE Name and an external entity does not necessarily carry over during matching.

7.1. Conceptual Model

The conceptual model for matching consists of two steps:

1. Make a list of all the CPE Names and logical connections.

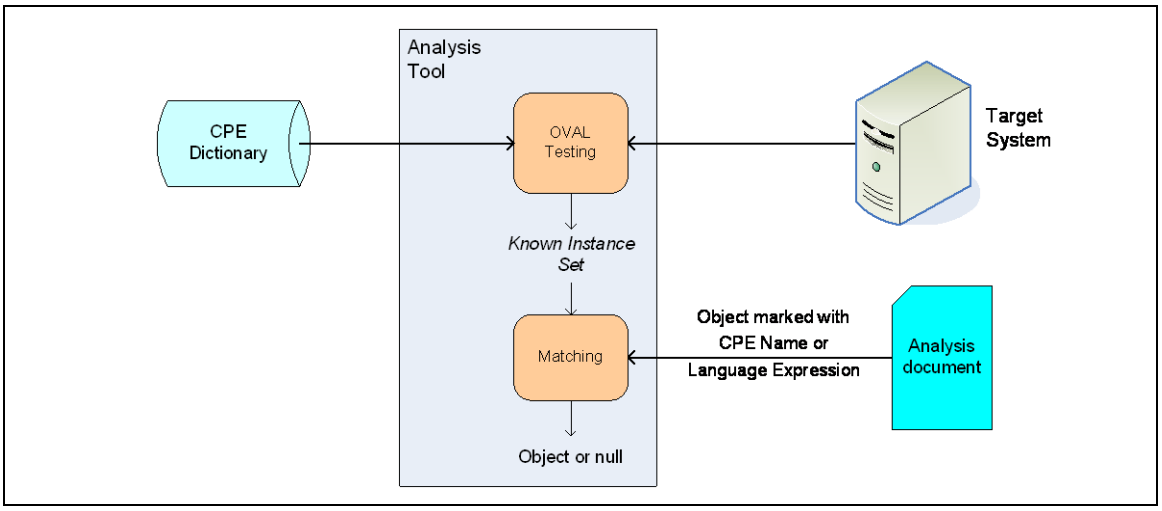
2. For each name, check whether the target system has the hardware, software, or operating system indicated by the name. If the check succeeds for all names, and satisfies the

logical constraints, then the target is an instance of the CPE Name or Language representation.

While the conceptual model is very simple, it will not be practical for all analysis tools. It is very difficult, in general, to check the constituent parts of an IT system.

In more realistic situations, analysis tools will be supplied with standard dictionaries of CPE Names for common hardware, OS, and application products. Using these dictionaries, and the OVAL Definitions included in them, an analysis tool can assemble a known set ***K*** of CPE Names of which the target platform is known to be an instance. Then, given any specific CPE Name or Language expression, the tool can ascertain whether or not the target is an instance of it. This is called known instance based matching. Figure 3 illustrates the basic concept of known instance based matching; a more formal description is given below.

Figure 3 – Conceptual Model for Known Instance Based Matching



Note: there is no mechanism in CPE to stipulate platform qualification based on the absence of a system constituent. All matching is positive, not negative.

7.2. Matching Algorithm: Known Instance Based Matching

Matching consists of two algorithms.

1. CPE_Name_Match – this algorithm accepts a set of CPE Names *K* and a candidate CPE Name *X*. It returns true if *X* matches any member of *K*, and false otherwise.

2. CPE_Language_Match – this algorithm accepts a set of CPE Names *K* and a candidate expression *E*. It returns true if the expression can be satisfied by name matching against the members of *K*, and false otherwise.

The algorithms are expressed below in pseudo-code. Sample implementations of the algorithms will be provided separately at the CPE web site.

CPE Name Matching

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.

Algorithm:

```
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
            end
            if r = true then
                return true.
            end
        end
    end
    return false.
end
```

Notes:

- The function length(N) returns the number of components in a CPE Name N.
- The function comp(N,i) returns the i'th component of the CPE Name N as a string.

CPE Language Matching

Inputs:

- *K* – a set of *m* known CPE Names $K = \{K_1, K_2, \ldots, K_m\}$.
- *E* – an expression in the CPE Language, represented as the XML infoset for the platform element (see Section 6)

Output:

- True if *E* can be satisfied by language matching against *K*, false otherwise.

Algorithm:

```
function CPE_Language_Match(K, E)
    if element(E) = "platform" then
        for each C in children(E) do
            if element(C) = "logical-test" then
                return CPE_Language_Match(K, C).
            end
        end
    else if element(E) = "fact-ref" then
        return CPE_Name_Match(K, attribute(E, "name")).
    else if element(E) = "logical-test" then
        count := 0.
        len := 0.
        answer := false.
        for each C in children(E) do
            len := len + 1.
            if (CPE_Language_Match(K, C)) then
                count := count + 1.
            end
        end
        if attribute(E, "operator") = "AND" then
            if count = len then
                answer := true.
            end
        else if attribute(E, "operator") = "OR" then
            if count > 0 then
                answer := true.
            end
        end
        if attribute(E, "negate") = true then
            answer := ~answer.
        end
        return answer.
    else
        return false.
    end
end
```

Notes:

- The function element(E) returns the element name of an XML infoset node as a string.
- The function attribute(E, A) returns the value of E's attribute A as a string.

7.3. Matching Examples

The examples below illustrate known instance based matching.

In the first example, evaluating OVAL definitions from the CPE Dictionary against a target system results in two CPE Names. These CPE Names are collected into the set ***K*** of known instances.

K = {"cpe:/o:microsoft:windows_2000::sp3:pro", "cpe:/a:microsoft:ie:5.5"}

A rule in a security guidance checklist describes some settings to check on a system running Microsoft Windows 2000. The rule is marked with a CPE Name indicating this is called the candidate name *X*.

X = "cpe:/o:microsoft:windows_2000"

There are three components in *X*: C_1=o, C_2=microsoft, C_3=windows_2000. Stepping through the matching algorithm, we see that each component matches the corresponding component of the first CPE Name in *K*. So, the algorithm returns true and the rule can be applied to the target system.

In the second example, testing OVAL definitions from the CPE Dictionary against a target system results in two CPE Names.

K = {"cpe:/o:sun:sunos:5.9:::en-us", "cpe:/a:bea:weblogic:8.1",}

A rule in a security vulnerability report designates the vulnerability as applicable for BEA WebLogic application server 8.0 running on Solaris 8 or 9 (SunOS 5.8 or 5.9). This is represented by the following CPE Language statement:

X =
```
<cpe:platform id="123">
    <cpe:title>Sun Solaris 5.8 or 5.9 with BEA Weblogic 8.1 installed</cpe:title>
    <cpe:logical-test operator="AND" negate="FALSE">
        <cpe:logical-test operator="OR" negate="FALSE">
            <cpe:fact-ref name="cpe:/o:sun:solaris:5.8" />
            <cpe:fact-ref name="cpe:/o:sun:solaris:5.9" />
        </cpe:logical-test>
        <cpe:fact-ref name="cpe:/ a:bea:weblogic:8.1" />
    </cpe:logical-test>
</cpe:platform>
```

Stepping through the CPE Language Matching algorithm, we see that there are the top-level logical test has the operator "AND" and two clauses: one of them a nested logical test and the other a simple fact-ref. The nested logical test evaluates to true, because one of the two simple fact-refs matches a member of *K*. The other fact-ref is also true, because it also matches a member of *K*.

8. CPE Dictionary

The CPE Dictionary is the official collection of CPE Names. Its purpose is to provide a source of all known CPE Names as well as bind descriptive prose and diagnostic tests to a CPE Name. This metadata includes a title, notes, and an automated check to determine if a given platform matches the CPE Name.

The primary format of the CPE Dictionary is an XML document. The format is defined using a XML Schema[1]; the full schema appears in Appendix C. The schema defines two main elements:

- <cpe-list> – the root element and a container for multiple <cpe-item> elements.
- <cpe-item> – holds information about an individual CPE Name, including a title, a list of zero or more descriptive notes, a list of zero or more references, and zero or more diagnostic checks.

The CPE Dictionary may be used by IT testing and report generation tools, as well as APIs, web services, etc., to get friendly names and descriptions for the compact CPE Names used in machine-readable guidance items and vulnerability definitions.

Figure 4, below, shows a short sample CPE Dictionary.

Figure 4 – A Short CPE Dictionary

```
<?xml version="1.0">

<cpe-list xmlns="http://cpe.mitre.org/dictionary/2.0"
          xmlns:cpe_dict="http://cpe.mitre.org/dictionary/2.0">

   <cpe-item name="cpe:/o:redhat:enterprise_linux:3">
       <title>Red Hat Enterprise Linux 3</title>
   </cpe-item>

   <cpe-item name="cpe:/o:sun:sunos:5.8">
       <title>Sun Microsystems SunOS 5.8</title>
       <notes>
          <note>Also known as Solaris 8</note>
       </notes>
   </cpe-item>

   <cpe-item name="cpe:/o:microsoft:windows_2003">
       <title>Microsoft Windows Server 2003</title>
       <check system="http://oval.mitre.org/XMLSchema/oval-definitions-5">
          oval:org.mitre.oval:def:128
       </check>
   </cpe-item>

</cpe-list>
```


8.1. Submissions to the CPE Dictionary

Vendors and community members can submit new names to the CPE Dictionary by emailing valid XML to cpe@mitre.org. The XML file should correspond to the dictionary schema that is outlined in Appendix C. Each submission can then be automatically imported into the official dictionary after a quick review process.

New submissions will be reviewed for correctness. This review will be used to make sure proposed names follow the guidelines set forth in this specification. Every effort will be made to make sure this process happens in the least amount of time possible.

Internal Process

- receive submission file from community
- validate the XML file
    - if it fails validation, notify user and try to correct issues
- determine if the file contains new or modified submissions
- verify content is acceptable
- add valid new submissions to CPE Dictionary
- update CPE Dictionary with accepted modifications
    - deprecate any old CPE Names as necessary
- send email to community regarding status
    - how many of the submissions were added
    - how many of the submissions were rejected
        - and why

9. References

[1] Fallside, D.C., *XML Schema Part 0: Primer*, W3C Recommendation, 2 May 2001.

W3C documents are available from http://www.w3.org/.

[2] Berners-Lee, T., Fielding, R., and Masinter, L., *Uniform Resource Identifier (URI): Generic Syntax*, RFC 3986, January 2005.

Internet RFCs are available from http://www.rfc-editor.org.

[3] "OVAL – The Open Vulnerability and Assessment Language", The MITRE Corporation, September 2006.

This is the OVAL web site, http://oval.mitre.org/.

[4] Mockapetris, P., "Domain Names – Implementation and Specification", RFC 1035, November 1987.

Internet RFCs are available from http://www.rfc-editor.org.

[5] Grobauer, B., "CVE, CME, …CMSI? Standardising System Information", Siemens AG, April 2005.

This paper introduces the Common Model of System Information, a structured format for describing IT platforms.

[6] Phillips, A. and Davis, M., *Tags for Identifying Languages*, RFC 4646, September 2006.

Internet RFCs are available from http://www.rfc-editor.org.

Appendix A: CPE Name String Grammar

The BNF grammar below defines the syntax for CPE Name URIs.

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

component-list ::= type ":" vendor ":" product ":" version ":" update ":" edition ":" lang
                   type ":" vendor ":" product ":" version ":" update ":" edition
                   type ":" vendor ":" product ":" version ":" update
                   type ":" vendor ":" product ":" version
                   type ":" vendor ":" product
                   type ":" vendor
                   empty

type           ::= ( "a" | "h" | "o" )
                   empty

vendor         ::= string
                   empty

product        ::= string
                   empty

version        ::= string
                   empty

update         ::= string
                   empty

edition        ::= string
                   empty

lang           ::= langtag
                   empty


string         ::= ( ALPHA | DIGIT | PUNC ) +

PUNC           ::= ( "." | "_" | "-" | "~" | "%" )
```

Where ALPHA and DIGIT are defined in [2] and langtag is defined in [6].

Appendix B: CPE Language XML Schema

The schema below specifies the XML representation of the CPE Language.

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema targetNamespace="http://cpe.mitre.org/language/2.0"
            xmlns:cpe="http://cpe.mitre.org/language/2.0"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xml="http://www.w3.org/XML/1998/namespace"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">

    <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
                schemaLocation="http://www.w3.org/2001/xml.xsd"/>

    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            This XML Schema defines the CPE Language.  An individual CPE Name
            addresses a single part of an actual system.  To identify more
            complex platform types, there needs to be a way to combine
            different CPE Names using logical operators.  For example, there
            may be a need to identify a platform with a particular operating
            system AND a certain application.  The CPE Language exists to
            satisfy this need, enabling the CPE Name for the operating system
            to be combined with the CPE Name for the application.  For more
            information, consult the CPE Specification document.
        </xsd:documentation>
        <xsd:appinfo>
            <schema>CPE Language</schema>
            <author>Neal Ziring, Andrew Buttner</author>
            <version>2.2</version>
            <date>03/11/2009 09:00:00 AM</date>
        </xsd:appinfo>
    </xsd:annotation>

<!-- ================================================================== -->
<!-- ================================================================== -->
<!-- ================================================================== -->

    <xsd:element name="platform-specification">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                This element is the root element of a CPE Language XML
                documents and therefore acts as a container for child
                platform definitions.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="platform" type="cpe:PlatformType"
                             minOccurs="1" maxOccurs="unbounded" />
            </xsd:sequence>
        </xsd:complexType>

        <xsd:key name="platformKey">
            <xsd:selector xpath="cpe:platform"/>
            <xsd:field xpath="@id"/>
        </xsd:key>
    </xsd:element>
```



```
<!-- ==================================================================== -->
<!-- ============================  PLATFORM  ============================== -->
<!-- ==================================================================== -->

    <xsd:complexType name="PlatformType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The platform element represents the description or
                qualifications of a particular IT platform type. The platform
                is defined by the logical-test child element.    The id
                attribute holds a locally unique name for the platform.
                There is no defined format for this id, it just has to be
                unique to the containing language document.
            </xsd:documentation>
            <xsd:documentation xml:lang="en">
                The optional title element may appear as a child to a
                platform element. It provides a human-readable title for it.
                To support uses intended for multiple languages, this element
                supports the 'xml:lang' attribute. At most one title element
                can appear for each language.
            </xsd:documentation>
            <xsd:documentation xml:lang="en">
                The optional remark element may appear as a child of a
                platform element. It provides some additional description.
                Zero or more remark elements may appear. To support uses
                intended for multiple languages, this element supports the
                'xml:lang' attribute.  There can be multiple remarks for a
                single language.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="title"
                         type="cpe:TextType"
                         minOccurs="0"
                         maxOccurs=" unbounded" />
            <xsd:element name="remark"
                         type=" cpe:TextType"
                         minOccurs="0"
                         maxOccurs="unbounded" />
            <xsd:element name="logical-test"
                         type="cpe:LogicalTestType"
                         minOccurs="1"
                         maxOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:anyURI" use="required" />
    </xsd:complexType>

    <xsd:complexType name="LogicalTestType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The logical-test element appears as a child of a platform
                element, and may also be nested to create more complex
                logical tests. The content consists of one or more elements:
                fact-ref, and logical-test children are permitted. The
                operator to be applied, and optional negation of the test,
                are given as attributes.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="logical-test"
                         type="cpe:LogicalTestType"
                         minOccurs="0"
                         maxOccurs="unbounded" />
```



```
            <xsd:element name="fact-ref"
                         type="cpe:FactRefType"
                         minOccurs="0"
                         maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="operator" type="cpe:operatorEnumeration"
                       use="required" />
        <xsd:attribute name="negate" type="xsd:boolean"
                       use="required" />
    </xsd:complexType>

    <xsd:complexType name="FactRefType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The fact-ref element appears as a child of a logical-test
                element. It is simply a reference to a CPE Name that always
                evaluates to a Boolean result.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="name" type="cpe:namePattern" use="required" />
    </xsd:complexType>

<!-- ================================================================= -->
<!-- =========================  ENUMERATIONS  ========================== -->
<!-- ================================================================= -->

    <xsd:simpleType name="operatorEnumeration">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The OperatorEnumeration simple type defines acceptable
                operators. Each operator defines how to evaluate multiple
                arguments.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="AND" />
            <xsd:enumeration value="OR" />
        </xsd:restriction>
    </xsd:simpleType>

<!-- ================================================================= -->
<!-- ========================  SUPPORTING TYPES  ====================== -->
<!-- ================================================================= -->

    <xsd:complexType name="TextType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                This type allows the xml:lang attribute to associate a
                specific language with an element's string content.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute ref="xml:lang" />
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

<!-- ================================================================= -->
<!-- =========================  ID PATTERNS  ========================== -->
<!-- ================================================================= -->

    <xsd:simpleType name="namePattern">
        <xsd:annotation>
```



```
            <xsd:documentation xml:lang="en">
                Define the format for acceptable CPE Names. A URN format is
                used with the id starting with the word cpe followed by :/
                and then some number of individual  components separated by
                colons.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:anyURI">
            <xsd:pattern value="[c][pP][eE]:/[AHOaho]?(:[A-Za-z0-9\._\-
~%]*){0,6}"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

Appendix C: CPE Dictionary XML Schema

The schema below specifies the XML format for the CPE Dictionary.

```
<?xml version="1.0"  encoding="UTF-8"?>

<xsd:schema targetNamespace="http://cpe.mitre.org/dictionary/2.0"
            xmlns:cpe_dict="http://cpe.mitre.org/dictionary/2.0"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xml="http://www.w3.org/XML/1998/namespace"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">

    <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
                schemaLocation="http://www.w3.org/2001/xml.xsd"/>

    <xsd:annotation>
       <xsd:documentation xml:lang="en">
           This is an XML Schema for the CPE Dictionary. It is used to
           transfer a collection of official CPE Names along with any
           necessary supporting information (title, references, automated
           check, etc.).  For more information, consult the CPE Specification
           document.
       </xsd:documentation>
        <xsd:appinfo>
            <schema>CPE Dictionary</schema>
            <author>Neal Ziring, Andrew Buttner</author>
            <version>2.2</version>
            <date>03/11/2009 09:00:00 AM</date>
        </xsd:appinfo>
    </xsd:annotation>

<!-- ================================================================= -->
<!-- ================================================================= -->
<!-- ================================================================= -->

    <xsd:element name="cpe-list" type="cpe_dict:ListType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The cpe-list element acts as a top-level container for CPE
                Name items.  Each individual item must be unique.  Please
                refer to the description of ListType for additional
                information about the sturcture of this element.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:key name="itemURIKey">
            <xsd:selector xpath="./cpe_dict:cpe-item"/>
            <xsd:field xpath="@name"/>
        </xsd:key>
    </xsd:element>

    <xsd:element name="cpe-item" type="cpe_dict:ItemType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The cpe-item element denotes a single CPE Name.  Please refer
                to the description of ItemType for additional information
                about the sturcture of this element.
            </xsd:documentation>
        </xsd:annotation>
```



```
            <xsd:unique name="titleLangKey">
                 <xsd:selector xpath="./cpe_dict:title"/>
                 <xsd:field xpath="@xml:lang"/>
            </xsd:unique>
            <xsd:unique name="notesLangKey">
                 <xsd:selector xpath="./cpe_dict:notes"/>
                 <xsd:field xpath="@xml:lang"/>
            </xsd:unique>
            <xsd:unique name="checkSystemKey">
                <xsd:selector xpath="./cpe_dict:check" />
                <xsd:field xpath="@system" />
            </xsd:unique>
        </xsd:element>

    <!-- ================================================================= -->
    <!-- ======================  SUPPORTING TYPES  ======================== -->
    <!-- ================================================================= -->

        <xsd:complexType name="GeneratorType">

            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                    The GeneratorType complex type defines an element that is
                    used to hold information about when a particular document was
                    compiled, what version of the schema was used, what tool
                    compiled the document, and what version of that tools was
                    used.  Additional generator information is also allowed
                    although it is not part of the official schema. Individual
                    organizations can place generator information that they feel
                    are important and these will be skipped during the
                    validation. All that this schema really cares about is that
                    the stated generator information is there.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="product_name" type="xsd:string"
                             minOccurs="0" maxOccurs="1">
                     <xsd:annotation>
                         <xsd:documentation xml:lang="en">
                             The optional product_name element specifies the name
                             of the application used to generate the file.
                         </xsd:documentation>
                     </xsd:annotation>
                </xsd:element>
                <xsd:element name="product_version" type="xsd:string"
                             minOccurs="0" maxOccurs="1">
                     <xsd:annotation>
                         <xsd:documentation xml:lang="en">
                             The optional product_version element specifies the
                             version of the application used to generate the
                             file.
                         </xsd:documentation>
                     </xsd:annotation>
                </xsd:element>
                <xsd:element name="schema_version" type="xsd:decimal"
                             minOccurs="1" maxOccurs="1">
                     <xsd:annotation>
                         <xsd:documentation xml:lang="en">
                             The required schema_version element specifies the
                             version of the schema that the document has been
                             written against and that should be used for
                             validation.
                         </xsd:documentation>
```



```
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="timestamp" type="xsd:dateTime"
                        minOccurs="1" maxOccurs="1">
                <xsd:annotation>
                    <xsd:documentation xml:lang="en">
                        The required timestamp element specifies when the
                        particular document was compiled. The format for the
                        timestamp is yyyy-mm-ddThh:mm:ss. Note that the
                        timestamp element does not specify item in the
                        document was created or modified but rather when the
                        actual XML document that contains the items was
                        created. For example, a document might pull a bunch
                        of existing items together, each of which having
                        been created at some point in the past. The
                        timestamp in this case would be when this combined
                        document was created.
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:any minOccurs="0" maxOccurs="unbounded"
                        namespace="##other" processContents="lax"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="ItemType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
                The ItemType complex type defines an element that represents
                a single CPE Name. The required name attribute is a URI which
                must be a unique key and should follow the URI structure
                outlined in the CPE Specification. The optional title element
                is used to provide a human-readable title for the platform.
                To support uses intended for multiple languages, this element
                supports the 'xml:lang' attribute.  At most one title element
                can appear for each language. The notes element holds
                optional descriptive material.  Multiple notes elements are
                allowed, but only one per language should be used.  Note that
                the language associated with the notes element applies to all
                child note elements.  The optional references element holds
                external info references.  The optional check element is used
                to call out an OVAL Definition that can confirm or reject an
                IT system as an instance of the named platform.  Additional
                elements not part of the CPE namespace are allowed and are
                just skipped by validation.  In essence, a dictionary file
                can contain additional information the a user can choose to
                use or not, but this information is not required to be used
                or understood.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="title" type="cpe_dict:TextType"
                        minOccurs="1" maxOccurs="unbounded"/>
            <xsd:element name="notes" type="cpe_dict:NotesType"
                        minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="references" type="cpe_dict:ReferencesType"
                        minOccurs="0" maxOccurs="1"/>
            <xsd:element name="check" type="cpe_dict:CheckType"
                        minOccurs="0" maxOccurs="unbounded"/>
            <xsd:any minOccurs="0"
                        maxOccurs="unbounded"
                        namespace="##other" processContents="lax"/>
        </xsd:sequence>
```



```
            <xsd:attribute name="name"
                           type="cpe_dict:namePattern"
                           use="required"/>
            <xsd:attribute name="deprecated"
                           type="xsd:boolean"
                           use="optional"
                           default="false"/>
            <xsd:attribute name="deprecated_by"
                           type="cpe_dict:namePattern"
                           use="optional"/>
            <xsd:attribute name="deprecation_date"
                           type="xsd:dateTime"
                           use="optional"/>
        </xsd:complexType>

        <xsd:complexType name="ListType">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                    The ListType complex type defines an element that is used to
                    hold a collection of individual items.  The required
                    generator section provides information about when the
                    definition file was compiled and under what version.
                    Additional elements not part of the CPE namespace are allowed
                    and are just skipped by validation.  In essence, a dictionary
                    file can contain additional information the a user can choose
                    to use or not, but this information is not required to be
                    used or understood.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="generator" type="cpe_dict:GeneratorType"
                             minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="cpe_dict:cpe-item"
                             minOccurs="1" maxOccurs="unbounded"/>
                <xsd:any minOccurs="0"
                         maxOccurs="unbounded"
                         namespace="##other" processContents="lax"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="TextType">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                    The TextType complex type allows the xml:lang attribute to
                    associate a specific language with an element's string
                    content.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                    <xsd:attribute ref="xml:lang" />
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>

         <xsd:complexType name="NotesType">
             <xsd:annotation>
                <xsd:documentation xml:lang="en">
                    The notesType complex type defines an element that consists
                    of one or more child note elements.  It is assumed that each
                    of these note elements are representative of the same
                    language as defined by their parent.
                </xsd:documentation>
```



```
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="note" type="xsd:string"
                             minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute ref="xml:lang"/>
        </xsd:complexType>

        <xsd:complexType name="ReferencesType">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                    The ReferencesType complex type defines an element used to
                    hold a collection of individual references.  Each reference
                    consists of a piece of text (intended to be human-readable)
                    and a URI (intended to be a URL, and point to a real
                    resource) and is used to point to extra descriptive material,
                    for example a supplier's web site or platform documentation.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="reference"
                             minOccurs="1" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="href" type="xsd:anyURI"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="CheckType">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                    The CheckType complex type is used to define an element for
                    hold information about an individual check.  It includes a
                    checking system specification URI, string content, and an
                    optional external file reference. The checking system
                    specification should be the URI for a particular version of
                    OVAL or a related system testing language, and the content
                    will be an identifier of a test written in that language. The
                    external file reference could be used to point to the file in
                    which the content test identifier is defined.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                  <xsd:attribute name="system" type="xsd:anyURI" use="required"/>
                  <xsd:attribute name="href" type="xsd:anyURI" use="optional" />
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>

    <!-- ================================================================== -->
    <!-- =========================  ID PATTERNS  ========================== -->
    <!-- ================================================================== -->

        <xsd:simpleType name="namePattern">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                    Define the format for acceptable CPE Names. A URN format is
```



```
                used with the id starting with the word cpe followed by :/
                and then some number of individual  components separated by
                colons.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:anyURI">
            <xsd:pattern value="[c][pP][eE]:/[AHOaho]?(:[A-Za-z0-9\._\-
~%]*){0,6}"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

