- Open Vulnerability and Assessment Language - Element Dictionary

- Schema: Core Definition
- Version: 4.2
- Release Date: 2 December 2005

The following is a description of the elements, types, and attributes that compose the core schema for encoding Open Vulnerability and Assessment Language (OVAL) Definitions. The Core Definition Schema defines all operating system independent objects. These objects are extended and enhanced by individual family schemas, which are described in separate documents. Each of the elements, types, and attributes that make up the Core Definition Schema are described in detail and should provide the information necessary to understand what each object represents. This document is intended for developers and assumes some familiarity with XML. A high level description of the interaction between these objects is not outlined here.

The OVAL Schema is maintained by The Mitre Corporation and developed by the public OVAL Community. For more information, including how to get involved in the project and how to submit change requests, please visit the OVAL website at http://oval.mitre.org.

Elements

This section describes all the elements that are found within the schema, starting with the root element. Note that in the tables outlining possible attributes and child elements, square brackets [] means that the item is optional. All complex and simple types, along with attribute groups are described later in this document.

<oval>

The oval element is the root of an OVAL Definition Document, and must occur exactly once. Its purpose is to bind together the four major sections of a definition - generator, definitions, tests, and variables - which are the children of the oval element. The generator section must be present and provides information about when the definition file was compiled and under what version. The optional definitions, tests, and variables sections define the specific characteristics that should be checked on a system to determine the truth value of the OVAL Definition Document.

The optional Signature element allows an XML Signature as defined by the W3C to be attached to the document. This allows authentication and data integrety to be provided to the user. Enveloped signatures are supported.

Cardinality:	1
Attributes:	none

Content:	none
Parent Elements:	none
Child Elements:	generator, [definitions], [tests], [variables], [Signature]

<generator>

The generator element is used to format information about when a particular OVAL Definition Document was compiled and what version of the schema was used. Note that the timestamp does not specify when a definition (or set of definitions) was created or modified.

Cardinality:	1
Attributes:	none
Content:	none
Parent Elements:	oval
Child Elements:	schema_version, timestamp

<schema_version>

The schema_version element defines the version of the OVAL Schema that the document has been validated against.

Cardinality:	1
Attributes:	none
Content:	decimal
Parent Elements:	generator
Child Elements:	none

<timestamp>

The timestamp element specifies the date/time at which the OVAL Definition Document was compiled. Note that the timestamp does not specify when a definition (or set of definitions) was created or modified. This timestamp can be used to differentiate between multiple OVAL files and to determine which document is the most up-to-date. The timestamp is a string in the form yyyymmddhhmmss.

Cardinality:	0-1
Attributes:	none
Content:	string

Parent Elements:	generator
Child Elements:	none

<definitions>

The definitions element is a container for one or more definition child elements.

Cardinality:	0-1
Attributes:	none
Content:	none
Parent Elements:	oval
Child Elements:	definition

<definition>

The definition element defines an actual OVAL Definition. It contains various metadata related child elements which describe the definition. This includes valid platforms, creation and modification dates, status information, and reference information. It also (unless the definition is deprecated) contains a criteria child element which joins individual tests together with a logical operator to specify the specific computer state being described.

The required id attribute is the OVAL-ID of the Definition. It has the form 'OVAL' followed by a number of digits (e.g. OVAL96). Like all ids in OVAL, it must be unique. OVAL-IDs are assigned by MITRE. The required class attribute indicates the specific class to which the definition belongs. Possible classes are: compliance, deprecated, patch, and vulnerability.

A definition is the key structure in OVAL. It is analogous to a logical sentence or proposition: if a computer's state matches the configuration parameters laid out in the criteria, then that computer exhibits the state described.

Cardinality:	1-n
Attributes:	id, class
Content:	none
Parent Elements:	definitions
Child Elements:	[affected], dates, description, [reference], status, version, [notes], [criteria]

<affected>

Each OVAL Definition is written to evaluate a certain type of system. The family, platform(s), and

product(s) of this target are described in the affected element whose main purpose is to provide hints for tools using OVAL Definitions. For instance, to help keep Windows definitions from being needlessly evaluated on a Red Hat machine. Note, the inclusion of a particular platform or product does not mean the definition is physically checking for the existence of the platform or product. For the actual test to be performed, the correct test must still be included in the definition's criteria section.

The required family attribute states the major category of operating system for which the definition is written. Each family has a corresponding family-specific definition schema which extends the Core Definition Schema.

Cardinality:	0-1
Attributes:	family
Content:	none
Parent Elements:	definition
Child Elements:	platform, [product]

<platformBase>

This abstract element details the specific platform(s) for which a definition has been written. It is extended by the individual family schemas to incorporate only the valid platforms for the specified family. The inclusion of a particular platform does not mean the definition is physically checking for the existence of the platform. For the actual test to be performed, the correct test must still be included in the definition’s criteria section. The valid platforms are outlined in the platform specific schemas.

Cardinality:	1-n
Attributes:	none
Content:	string
Parent Elements:	affected
Child Elements:	none

<product>

This element details the specific application, subsystem, library, etc. for which a definition has been written. If a definition is not tied to a specific product, then this element should not be included. The absence of the product element can be thought of as definition applying to all products. The inclusion of a particular product does not mean the definition is physically checking for the existence of the product. For the actual test to be performed, the correct test must still be included in the definition's criteria section. To increase the utility of this element, care should be taken when assigning and using strings for product names. The schema places no restrictions on the values that can be assigned, potentially leading to many different representations of the same value. For example 'Internet Explorer' and 'IE'. The current convention is to fully spell out all terms, and avoid the use of abbreviations at all costs.

Cardinality:	0-n
Attributes:	none

Content:	string
Parent Elements:	affected
Child Elements:	none

<dates>

This element contains child elements to hold submission, modification, and status change dates associated with the definition.

Cardinality:	1
Attributes:	none
Content:	none
Parent Elements:	definition
Child Elements:	submitted, [modified], [status_change]

<submitted>

This element identifies when a definition was submitted to the OVAL Community. A definition can only be submitted once. As children, an unbounded number of optional contributor elements outline who is credited with the submission. The required date attribute holds that actual date of the submission. It is of type date and should be of the form yyyy-mm-dd.

Cardinality:	1
Attributes:	date
Content:	none
Parent Elements:	dates
Child Elements:	[contributor]

<modified>

This element identifies when a definition was modified and provides details about what modification was made. A definition can be modified an unlimited number of times. The unbounded number of optional child contributor elements outline who is credited with the modification. The required date attribute identifies when the change was actually made. It is of type date and should be of the form yyyy-mm-dd.

Cardinality:	0-n
Attributes:	date, comment
Content:	none
Parent Elements:	dates
Child Elements:	[contributor]

<status_change>

This element identifies when an OVAL Definition changed status. This represents the movement of a definition through the review process. This is an automatic change and not associated with an individual. The required date attribute identifies when the change was actually made. It is of type date and should be of the form yyyy-mm-dd.

Cardinality:	0-n
Attributes:	date
Content:	string
Parent Elements:	dates
Child Elements:	none

<contributor>

The contributor element identifies by name the member of the OVAL Community who is credited with a particular submission or modification. The optional organization attribute identifies the organization with which the contributor is affiliated.

Cardinality:	1
Attributes:	[organization]
Content:	none
Parent Elements:	modified, submitted
Child Elements:	none

<description>

The description element contains a textual description of the configuration state being addressed by the OVAL Definition. In the case of a definition from the vulnerability class, the reference is usually the Common Vulnerability and Exposures (CVE) Identifier, and this description field corresponds with the CVE description.

Cardinality:	1
Attributes:	none
Content:	string
Parent Elements:	definition
Child Elements:	none

<reference>

This element links the OVAL Definition to a definitive external reference. For example, CVE Identifiers for vulnerabilities. The intended purpose for this reference is to link the definition to a variety of other sources that share this common name or identifier. Only one reference is allowed for each definition and the required source attribute serves as an indicator to the reference type.

Cardinality:	0-n
Attributes:	source
Content:	string
Parent Elements:	definition
Child Elements:	none

<status>

This element contains the current status of the definition. Possible values are: ACCEPTED, DEPRECATED, DRAFT, INCOMPLETE, INITIAL SUBMISSION, and INTERIM. Status changes are managed by MITRE. Please visit the OVAL website at http://oval.mitre.org for more information about the each status.

Cardinality:	1
Attributes:	none
Content:	string
Parent Elements:	definition
Child Elements:	none

<version>

This element holds the current version of the definition. Versions are integers, starting at 0 and incrementing every time a definition reaches ACCEPTED status. For example, an ACCEPTED definition with a version of 1 that is modified will return to INTERIM status for some period of time while the modifications are reviewed. During this period the definition will maintain a version of 1. When the changes have been approved, the definition status will become ACCEPTED again, but the will now have a version of 2.

Cardinality:	1
Attributes:	none
Content:	integer
Parent Elements:	definition
Child Elements:	none

<notes>

This element is a container for one or more note child elements.

Cardinality:	0-1
Attributes:	none
Content:	none
Parent Elements:	definition, test
Child Elements:	note

<note>

A note contains some descriptive text about the containing definition or individual test element. It may record an unresolved question about the definition or test, document an unknown_test, or present the reasoning as to why a particular approach was taken.

Cardinality:	1-n
Attributes:	none
Content:	string
Parent Elements:	notes
Child Elements:	none

<criteria>

Each definition is described by a number of tests, referenced by the individual criterion elements. The criteria element is the high level container for all the tests and represents the meat of the definition. These tests are broken up into two different categories, software and configuration. This categorization allows users to differentiate between the 'software on disk' portion of a definition and the 'how is the machine configured' portion of the definition.

Cardinality:	0-1
Attributes:	none
Content:	none
Parent Elements:	definition
Child Elements:	[software], [configuration]

<software>

Software tests describe specific conditions that exist with software on disk. For example, an OVAL definition might address a vulnerability that exists in a Microsoft Windows .dll file. The conditions about whether this .dll file actually exists on a machine are outlined in the software section. The optional operation attribute determines how to handle multiple criterion elements. Possible values are: AND, OR, XOR. A value of AND means that each criterion must be true for the software section to return true. A value of OR means that only one criterion must be true for the software section to return turn. XOR is defined to be true if an odd number of criterion are true, and false otherwise.

Cardinality:	0-1
Attributes:	[operation]
Content:	none
Parent Elements:	criteria
Child Elements:	criterion

<configuration>

Tests under the configuration section describe conditions that violate a secure baseline or recommendation. For example, if a vulnerable Microsoft Windows .dll file is part of a specific service, then a machine might only be exploitable if that service is actually running. Even though the vulnerable file exists on the disk, the vulnerability can be mitigated by disabling the service. The test determining if the service is running would be under the configuration section. The optional operation attribute determines how to handle multiple criterion elements. Possible values are: AND, OR, XOR. A value of AND means that each criterion must be true for the configuration section to return true. A value of OR means that only one criterion must be true for the configuration section to return turn. XOR is defined to be true if an odd number of criterion are true, and false otherwise.

Cardinality:	0-1
Attributes:	[operation]
Content:	none
Parent Elements:	criteria
Child Elements:	criterion

<criterion>

The criterion element identifies a specific test to be included in either the software or configuration section of a definition's criteria. The required test_ref attribute is the actual id of the test being referenced. The optional negate attribute signifies that the result of an individual test should be negated during analysis. For example, consider a test that returns TRUE if a specific patch is installed. By negating this test, it now analyzes to TRUE if the patch is NOT installed. The required comment attribute provides a short description of the specified test and should mirror the comment attribute of the actual test.

Cardinality:	1-n
Attributes:	test_ref, [negate], comment
Content:	none
Parent Elements:	software, configuration
Child Elements:	none

<tests>

This element is a container for one or more test child elements.

Cardinality:	0-1
Attributes:	none
Content:	none
Parent Elements:	oval
Child Elements:	test

<test>

This is an abstract element that is meant to be extended (via substitution groups) by the tests found in the family schemas. An actual test element is not valid. The use of this abstract class simplifies the OVAL schema and allows descriptive element names to be used in place of test. The abstract test element inherits the optional notes child element, and the id and comment attributes from the base testType. A description of the notes element can be found under the definitions section. Please refer to the "Complex Types" section of this document for a description of the testType.

Cardinality:	1-n
Attributes:	id, comment
Content:	none
Parent Elements:	tests
Child Elements:	[notes], (specified through extension)

<compound_test>

This test has been deprecated in version 4.1 of the oval-schema and will be removed completely in version 5. It is recommended that all future OVAL Content use the compound_test found in the independent schema.

A compound test allows multiple tests (including other compound tests) to be joined together by a logical operator. This provides flexibility in test creation and enables complex tests to be reused, serving as building blocks for future tests. The required operation attribute specifies how to logically combine the numerous subtests of a compound test. Possible values are: AND, OR, XOR. A value of AND means that each subtest must be true for the compound_test to return true. A value of OR means that only one subtest must be true for the compound_test to return true. A value of XOR means that one, and only one, subtest must be true for the compound_test to return true. A compound test extends the testType. Please refer to the "Complex Types" section of this document for a description of the testType.

Extends:	testType
Valid Sections:	[notes], subtest

```
<compound_testid="cmp-0"operation="AND"comment="an example compound test">
        <oval:notes>
                <oval:note>This is an example test written under version 4 of the OVAL
                schema. It ANDs together the results of three separate tests, one of which is
                negated.</oval:note>
        </oval:notes>
        <subtesttest_ref="wrt-0"/>
        <subtesttest_ref="wat-0"negate="true"/>
        <subtesttest_ref="cmp-1"/>
</compound_test>
```

<subtest>

The subtest element specifies a particular test to be referenced. The required test_ref attribute accomplishes this by linking to a valid test id. The optional 'negate' attribute signifies that the result of an individual test should be negated during analysis. For example, consider a test that returns TRUE if a specific patch is installed. By negating this test, it now analyzes to TRUE if the patch is NOT installed.

Parent Test:	Compound Test
Cardinality:	1-n
Content:	none

<unknown_test>

This test has been deprecated in version 4.1 of the oval-schema and will be removed completely in version 5. It is recommended that all future OVAL Content use the unknown_test found in the independent schema.

An unknown test acts as a placeholder for tests whose implementation is unknown. Any information that is known about the test should be held in the notes child element that is available through the extension of the abstract test element. An unknown test extends the testType. Please refer to the "Complex Types" section of this document for a description of the testType.

Extends:	TestType
Valid Sections:	[notes]

```
<unknown_testid="ukn-0"comment="an example unknown test">
        <oval:notes>
                <oval:note>This is an example test written under version 4 of the OVAL
                schema. A description about the desired test would go here including what is
                unknown about it.</oval:note>
        </oval:notes>
</unknown_test>
```

<variable_test>

This test has been deprecated in version 4.1 of the oval-schema and will be removed completely in version 5. It is recommended that all future OVAL Content use the variable_test found in the independent schema.

A variable test allows the value of a variable to be compared to a defined value. An example use would be to validate that a variable being passed in from an external source falls within a specified range.

Extends:	TestType
Valid Sections:	[notes], item

```
<variable_testid="vct-0"operation="AND"comment="an example variable test">
        <itemvariable="var-3"datatype="int"operator="greater than">6</item>
        <itemvariable="var-3"datatype="int"operator="less than"var_ref="var-6"/>
</variable_test>
```

<variables>

This element is a container for one or more variable child elements.

Cardinality:	0-1
Attributes:	none
Content:	none
Parent Elements:	oval
Child Elements:	variable

<variable>

A variable element is a reference to some value that can be used by tests to compare against. One of the main benefits of variables is that they allow tests to be compared to user-defined policy. For example, an OVAL test might check to see if a password is at least a certain number of characters, but this number depends upon the individual policy of the user. To solve this, the test for password length can be written to refer to a variable element. This variable element refers to an external value (linked by the id) that can be passed in by the user when the OVAL definition is evaluated. The required id attribute uniquely identifies each variable. It is of the form var-#. The three letter code 'var' is followed by an unspecified number of digits, for example 'var-123'. The required datatype attribute specifies the type of value to expect in the external source. The required source attribute determines where the value of the variable will be found, either from some external source or as a constant declaration. When the source of the value is external, the id is used to link the variable element to the external source of the value. The required comment attribute provides a short description of the variable.

Cardinality:	1-n
Attributes:	id, datatype, source, comment
Content:	none
Parent Elements:	variables
Child Elements:	[restricted], [value]

<restricted>

This element is a container of 'possible' child elements that specify any restrictions on the values of a variable. This element should only be available when the source attribute of a variable is 'external', although this is not enforced by this schema.

Cardinality:	0-1
Attributes:	none
Content:	none
Parent Elements:	variable
Child Elements:	possible

<possible>

This element outlines a possible expected value of a variable. A value that is passed in from an external source must match one of the possible values if the variable is being restricted. The required hint attribute gives a short description of what the value means. The required operator attribute specifies how to compare the actual value of the variable with the possible value.

Cardinality:	0-1
Attributes:	hint, operator
Content:	** undefined **
Parent Elements:	restricted
Child Elements:	** undefined **

<value>

If the source of a variable is constant, then the 'value' child element holds the actual value of the variable. NOTE that this element should only be (and must be) present when the source of a variable is 'constant', even though there is no way to enforce this during validation. This value should be used by all tests that reference this variable. The value can not be over-ridden by an external source.

Cardinality:	0-1
Attributes:	none

Content:	** undefined **
Parent Elements:	variable
Child Elements:	** undefined **

Complex Types

This section describes any global complex types defined in the schema. These types can be instantiated by elements in this schema as well as elements in other schemas. Note that in the tables outlining possible attributes and child elements, square brackets [] means that the item is optional.

-- testType --

The base type of every test includes an optional notes element and two attributes. The required id attribute uniquely identifies each test, and is of the form 'xxx-#'. There is a three letter character code to help distinguish the type of test, followed by an unspecified number of digits (e.g. 'wrt-123'). The required comment attribute provides a short description of the test.

Attributes:	id, comment
Content:	none
Child Elements:	[notes]

-- standardTestType --

The standardTestType is an extension of the testType. The optional check attribute determines what group of objects to test. (For example: Should the test check that all files match a specified version or that at least one file matches the specified version?) The standardTestType is extended by individual tests in the different family schemas.

Extends:	testType
Attributes:	[check]
Content:	none
Child Elements:	none

-- objectType --

The objectType is extended by the individual tests found in the different family schemas. The object section contains the child elements that describe which objects to gather data about and analyze.

Attributes:	none
Content:	none
Child Elements:	none

-- dataType --

The dataType is extended by the individual tests found in the different family schemas. The data section contains the set of child elements whose values represent the testable parameters for the specified objects. The optional operation element defines the logical relation between multiple elements of the data section.

Attributes:	[operation]
Content:	none
Child Elements:	none

-- subtestBoolType --

The subtestBoolType type is extended by the child elements of an individual test. This type provides uniformity to each child element by including the attributes found in the subtestAttributes group. This attribute is included by other subtest types and makes the same list of attributes available to all children of a test. This specific type describes simple boolean data.

Attributes:	(includes subtestAttributes)
Content:	boolean
Child Elements:	none

-- subtestIntType --

The subtestIntType type is extended by the child elements of an individual test. This type provides uniformity to each child element by including the attributes found in the subtestAttributes group. This attribute is included by other subtest types and makes the same list of attributes available to all children of a test. This specific type describes simple integer data.

Attributes:	(includes subtestAttributes)
Content:	integer
Child Elements:	none

-- subtestStringType --

The subtestStringType type is extended by the child elements of an individual test. This type provides uniformity to each child element by including the attributes found in the subtestAttributes group. This attribute is included by other subtest types and makes the same list of attributes available to all children of a test. This specific type describes simple string data.

Attributes:	(includes subtestAttributes)
Content:	string
Child Elements:	none

-- subtestBaseType --

The subtestBaseType type is extended by the child elements of an individual test. This type provides uniformity to each child element by including the attributes found in the subtestAttributes group. This attribute is included by other subtest types and makes the same list of attributes available to all children of a test. This specific type describes complex data.

Attributes:	(includes subtestAttributes)
Content:	(anyType)
Child Elements:	(anyType)

Attribute Groups

This section describes any global attribute groups defined in the schema. An attribute group can be included by various types providing a standard set of attributes across each of the types. Note that in the tables outlining possible attributes, square brackets [] means that the item is optional.

-- subtestAttributes --

The following are the default attributes associated with every test element's children. The optional datatype determines the type of data expected. (the default datatype is 'string') The optional operator determines how the individual test cases (the child elements) should operate. (the default operator is 'equals') Both of these attributes are optional in order to keep the XML clean and readable. The default values are used most of the time and putting datatype="string" and operator="equals" for each element would muddy up the XML. The optional var_ref attribute refers the value of the child to a variable element. Some thought was given to removing the datatype attribute but it was decided that it is needed to determine how to programmatically perform the operation. For example the less than operator against '41' and '9' gives different results if doing a string compare or an integer compare. Use of the xsi:type attribute was considered, but it was decided that this would be too complex and put too much responsibility on the user to chose the correct type for a given subtest element.

Attributes:	[datatype], [operator], [var_ref]

Simple Types

This section describes any global simple type defined in the schema. A simple type is a restriction of one of the base types (string, int, etc.) and allows a valid entry to be limited to a specific subset of values.

---

check values

The check type defines acceptable check values. A value of 'all' means that all matching objects must satisfy data requirements for a test to return true. 'at least one' means that at least one matching object must satisfies the data requirements. 'none exists' means that a test returns true if no matching object exist that satisfy the data requirements. 'only one' means that a test returns true if one, and only one, matching object satisfies the data requirements.

-- all
-- at least one
-- none exist
-- only one

datatypes values

This simple type defines the legal datatypes that are used to describe the values of a test's child elements. A value should be interpreted according to the specified type. This is most important during comparisons. For example, "Is '21' less than '123'?" will evaluate to true if the datatypes are 'int, but will evaluate to 'false' if the datatypes are 'string'.

The 'binary' datatype is used to represent data that is in raw (non-printable) form. Values should be hex strings. The 'boolean' datatype describes true or false values. The strings 'true' and 'false' are acceptable values, as are the numbers 1 and 0. The 'float', 'int', and 'string' datatypes are used to describe data of these types.

The component datatype represents a string value that is built from one or more component strings. Each component string is concatenated together to form the final string used by the element. The individual components can be a literal string or can a value returned from some another source, for example a registry key. If the source does not exist, i.e. the registry can not be found, then an error should be reported.

The rpmversion datatype represents the version of a redhat rpm. It is a string with the form epoch:version-release. If epoch is null then the string NULL is used. For example, NULL:0.9.13-1.90.1

The version datatype represents a value that is a hierarchical list of versions. For example '#.#.#' or '#-#-#-#' where the numbers to the left are more significant than the numbers to the right. When performing an 'equals' operation on a version datatype, you should first check the left most number for equality. If that fails, then the values are not equal. If it succeeds, then check the second left most number for equality. Continue checking the numbers from left to right until the last number has been checked. If, after testing all the previous numbers, the last number is equal then the two versions are equal. When performing other operations, such as 'less than', 'less than or equal', 'greater than, or 'greater than or equal', similar logic as above is used. Start with the left most number and move from left to right. For each number, check if it is less than the number you are testing against. If it is, then the version in question is less than the version you are testing against. If the number is equal, then move to check the next number to the right. For example, to test if 5.7.23 is less than or equal to 5.8.0 you first compare 5 to 5. They are equal so you

move on to compare 7 to 8. 7 is less than 8 so the entire test succeeds and 5.7.23 is 'less than or equal' to 5.8.0. The difference between the 'less than' and 'less than or equal' operations is how the last number is handled. If the last number is reached, the check should use the given operation (either 'less than' and 'less than or equal') to test the number. For example, to test if 4.23.6 is greater than 4.23.6 you first compare 4 to 4. They are equal so you move on to compare 23 to 23. They are equal so you move on to compare 6 to 6. This is the last number in the version and since 6 is not greater than 6, the entire test fails and 4.23.6 is not greater than 4.23.6.

-- binary
-- boolean
-- component
-- float
-- int
-- rpmversion
-- string
-- version

definitionclass values

The different classes of definitions. A compliance definition describes the state of a machine as it complies with a specific policy. A patch definition details the machine state of whether a patch should be installed. A vulnerability definition described the condition under which a machine is vulnerable. A deprecated definition is placeholder for an OVAL definition that was officially accepted but has since been removed.

-- compliance
-- deprecated
-- patch
-- vulnerability

definitionid values

Define acceptable OVAL names as the string 'OVAL', followed by some number of digits.

-- a value satisfying the pattern '(OVAL[0-9]+)|(oval:[A-Za-z\-\.]+:def:[1-9][0-9]*)'

families values

The families simple type is a listing of families that OVAL supports at this time.

-- aix
-- apache
-- debian
-- freebsd
-- hp-ux
-- ios
-- macos
-- openbsd
-- oracle
-- os400

-- pix
-- redhat
-- solaris
-- suse
-- windows

operations values

Define acceptable operations. XOR is defined to be true if an odd number of its arguments are true, and false otherwise.

-- AND
-- OR
-- XOR

operators values

Define acceptable operators.

-- equals
-- not equal
-- greater than
-- less than
-- greater than or equal
-- less than or equal
-- bitwise and
-- bitwise or
-- pattern match

reference_source values

The different sources for a reference

-- CVE
-- MISC

status values

The status of an OVAL definition.

-- ACCEPTED
-- DEPRECATED
-- DRAFT
-- INCOMPLETE
-- INITIAL SUBMISSION
-- INTERIM

testid values

Define acceptable test ids as a three character string followed by a hyphen and some number of digits.

-- a value satisfying the pattern '([a-z]{3}-[0-9]+)|(oval:[A-Za-z\-\.]+:tst:[1-9][0-9]*)'

timestamp values

Define acceptable timestamps as a string with the form yyyymmddhhmmss.

-- a value satisfying the pattern '\d{14}'

variable_source values

The different sources for a variable value. An external source means the value is retrieved from somewhere outside of OVAL, say a variable file or directly from the analysis code. Think of this as when a value is passed into OVAL. A constant source means the value is declared inside of OVAL and can not be modified.

-- constant
-- external

varid values

Define acceptable variable ids as the string 'var-' followed by some number of digits.

-- a value satisfying the pattern '(var-[0-9]+)|(oval:[A-Za-z\-\.]+:var:[1-9][0-9]*)'