- Open Vulnerability and Assessment Language - Element Dictionary

- Schema: Core Results
- 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) Results. The Core Results 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 Results 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_results>

The root element of an OVAL Results Document binding all of the results together along with the associated test elements. It must contain exactly one generators child element and exactly one system_info child element. The other child elements are optional.

The oval_results element is the root of an OVAL results document, and must occur exactly once. Its purpose is to bind together the five major sections of a results file - generators, system_info, definitions, tests, and variables - which are the children of the oval_results element. The generator section must be present and provides information about when the results file was compiled and under what version. The require system_info element is used to record information about the system being described.

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:	generators, system_info, [definitions], [tests], [variables], [Signature]

<generators>

The generators element specifies information about who generated the set of OVAL definitions used in the analysis, what application collected the data used for the analysis, and what application preformed the analysis. Note that the system_characteristics generator information is optional to allow for other data sources.

Cardinality:	1
Attributes:	none
Content:	none
Parent Elements:	oval_results
Child Elements:	oval, system_characteristics, results

<oval>

Specifies the source of the OVAL Definitions.

Cardinality:	1
Attributes:	none
Content:	none
Parent Elements:	generators
Child Elements:	product_name, product_version, schema_version, timestamp

<system_characteristics>

Specifies the application used for data collection.

Cardinality:	0-1
Attributes:	none
Content:	none

Parent Elements:	generators
Child Elements:	product_name, product_version, schema_version, timestamp

<results>

Specifies the application used for analysis.

Cardinality:	1
Attributes:	none
Content:	none
Parent Elements:	generators
Child Elements:	product_name, product_version, schema_version, timestamp

<product_name>

The name of the application used to generate this file of OVAL definitions.

Cardinality:	0-1
Attributes:	none
Content:	string
Parent Elements:	oval, system_characteristics, results
Child Elements:	none

<product_version>

The version of the product used to generate this file of OVAL definitions

Cardinality:	0-1
Attributes:	none
Content:	string
Parent Elements:	oval, system_characteristics, results
Child Elements:	none

<schema_version>

This element defines the version of the OVAL schema that the document has been validated against.

Cardinality:	1
Attributes:	none
Content:	decimal
Parent Elements:	oval, system_characteristics, results
Child Elements:	none

<timestamp>

This element specifies the date/time at which the document was created. 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 of the form yyyymmddhhmmss.

Cardinality:	1
Attributes:	none
Content:	string
Parent Elements:	oval, system_characteristics, results
Child Elements:	none

<system_info>

The system_info element specifies general information about the system that was analyzed including information that can be used to identify the system.

Cardinality:	1
Attributes:	none
Content:	string
Parent Elements:	system_characteristics
Child Elements:	os_name, os_version, architecture, primary_host_name, interfaces

<os_name>

The operating system of the machine that was analyzed.

Cardinality:	1
Attributes:	none
Content:	string

Parent Elements:	system_info
Child Elements:	none

<os_version>

The version of the operating system of the machine that was analyzed.

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

<architecture>

The architecture of the machine the that was analyzed.

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

<primary_host_name>

The primary host name of the machine that was analyzed.

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

<interfaces>

The interfaces element holds a collection of interface elements that describe each interface on the machine that was analyzed.

Cardinality:	1

Attributes:	none
Content:	string
Parent Elements:	system_info
Child Elements:	interface

<interface>

The interface element is used to describe an interface on a system.

Cardinality:	1-n
Attributes:	none
Content:	string
Parent Elements:	interfaces
Child Elements:	interface_name, ip_address, mac_address

<interface_name>

The name of the interface

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

<ip_address>

The ip address of the interface

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

<mac_address>

The mac address of the interface

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

<definitions>

The definitions element is a container for the results of the analysis on each OVAL definition specified.

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

<definition>

The definition element describes meta-data for an the analysis results of an individual OVAL definition. It also contains a criteria element that references the individual tests that make up the definition.

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 optional instance identifier is a unique id that differentiates every unique instance of a definition in the oval results file. Languages that include OVAL might reference the same definition multiple times. Each time a different set of values is supplied for the variables, a new instance of the definition must be created. (definitions that do not use variables can only have one unique instance) The inclusion of a unique instance identifier will allow the OVAL results file to report the correct result of a definition for each combination of supplied values. The required class attribute indicates the specific class the definition belongs to. Possible classes are: compliance, deprecated, patch, vulnerability.

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

<affected>

Each OVAL definition is written for a particular system and the target family, platform(s), and product(s) are described in the affected element. The affected element's main purpose is to provide hints for tools using OVAL definitions, e.g. so Windows definitions are not needlessly evaluated on a Red Hat machine. 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 preformed, the correct test must still be included in the definition's criteria section.

The family attribute states which major category of operating system the definition is written for. Possible values are: debian, ios, redhat, solaris, or windows. More families will be added to OVAL as needed. Each family has a corresponding family-specific definition schema which extends this core OVAL Definition schema.

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

<platformBase>

This element details a specific platform that the definition has been written for. 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 preformed, 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 a specific application, subsystem, library, etc. that the definition has been written for. 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 preformed, the correct test must still be included in the definition's criteria section.

Cardinality:	0-n
Attributes:	none
Content:	string

Parent Elements:	affected
Child Elements:	none

<description>

This element contains a textual description of what the OVAL definition is testing for. In the case of a vulnerability class definition that references a CVE entry, it is recommended that the description is the same as 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:	0-1
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, INTERIM. Please refer to the oval website for a description of each of these statuses. Status changes are managed by MITRE.

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 (ie, an ACCEPTED version 1 definition which requires modification will return to INTERIM status, remaining at version 1, and will move to version 2 when the changes have been approved and it has become ACCEPTED again).

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

The optional result attribute holds the result of the analysis as a whole: either true, false, or error. This result assumes that both the software and configuration section are used in the determination. If this is not the case, then this result should be left out. Some tools will only evaluate the software section, or only evaluate the configuration section, and would then rely solely on the result attribute associated with that section.

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

<software>

Software tests describe specific conditions that exists with software on disk. For example, an OVAL definition might talk about a vulnerability that exists in a .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. A value of XOR means that one, and only one, criterion must be true for the software section to return true. The required result attribute holds the result of the analysis, either true, false, or error.

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

<configuration>

Configuration tests describe conditions about whether something can actually be exploited due to how the machine is setup. For example, if a vulnerable .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 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. A value of XOR means that one, and only one, criterion must be true for the configuration section to return true. The required result attribute holds the result of the analysis, either true, false, or error.

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

<criterion>

This element specifies a specific tests to be included in either the software or configuration section of a definition's criteria. The required 'test_ref' attribute is actual id of the test being linked to. The optional version attribute signifies which version of the test was used during analysis. Different versions of a test can be encounted if a test contains a variable reference and different values for the variable are used in different definition instances. The optional 'negate' attribute signifies that a failed result of the test should be looked for instead of a successful result. For example, the test might normally return true if the patch is installed. By setting negate equal to true, what we get is the test element will return 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. The required result attribute holds the result of the analysis, either true, false, or error.

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

Child Elements:	none

<tests>

The tests element acts as a container for the detailed results of each test required by the specified OVAL definitions.

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

<test>

This is an abstract element that is meant to be extended (via substitution groups) by the different 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 id and comment attribute from the its base testType.

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

<message>

Holds a message from the analysis engine. For example, an error message.

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

<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. The required result attribute specifies the result of the OVAL analysis on this group of subtests. A compound test extends the testType.

Extends:	compoundTestType
Valid Sections:	[message], subtest

```
<compound_testid="cmp-0"operation="AND"comment="an example compound
test"version="1"result="1">
        <subtesttest_ref="wrt-0"version="1"result="0"/>
        <subtesttest_ref="wat-0"version="1"negate="true"result="1"/>
        <subtesttest_ref="cmp-1"version="1"result="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. The required result attribute holds the result of the analysis, either true, false, or error.

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. The required result attribute holds the result of the analysis, either true, false, or error.

Extends:	TestType
Valid Sections:	[message]

```
<unknown_testid="ukn-0"comment="an example unknown test"version="1"result="0">
</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:	[message], item

```
<variable_testid="vct-0"operation="AND"comment="an example variable
test"version="1"result="1">
        <itemvariable="var-3"version="1"datatype="int"operator="greater
        than"result="0">6</item>
        <itemvariable="var-3"version="1"datatype="int"operator="less
        than"result="1">78</item>
</variable_test>
```

<variables>

The variables element is a container for different values of the variables required by the specified OVAL definitions..

Cardinality:	0-1
Attributes:	none
Content:	none
Parent Elements:	oval_results
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 was be found, either from some external source or as a constant declaration. The required comment attribute provides a short description of the variable.

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

<value>

The 'value' element holds the actual value of the variable used during analysis.

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 a notes element and two attributes. The required id attribute uniquely identifies each test. It is of the form 'xxx-#'. The three letter character code helps distinguish the type of test. This is followed by an unspecified number of digits. For example 'wrt-123'. The optional version

attribute differentiates between different version of a test. This can happen when a test includes a variable reference and different values for that variable are used by different definitions. The required comment attribute provides a short description of the test.

Attributes:	id, comment, [version]
Content:	none
Child Elements:	[message]

-- 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 result attribute holds the result of the analysis, either true, false, or error. The standardTestType is extended by individual tests in the different family schemas.

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

-- definitionObjectType --

The objectType is extended by the individual tests found in the different family schemas. The object section contains the child elements that determine which objects to apply the test to.

Attributes:	none
Content:	none
Child Elements:	none

-- definitionDataType --

This dataType is extended by the individual tests found in the different family schemas. The data section contains the child elements that define the desired traits to test each object in the object section against. The optional operation element defines the logical relation between the multiple elements of the data.

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

-- testedObjectType --

This base type is extended by the individual tests found in the schemas of unique families. The

tested_object section contains the subtests that show which objects were tested on a system. The result attribute specified the result of the OVAL analysis on the specified object. The object_id is present to allow for a reference to an external document containing the data about the object.

Attributes:	result, [status], [object_id]
Content:	none
Child Elements:	none

-- subtestBoolType --

Describes simple boolean data along with the standard subtest attributes.

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

-- subtestIntType --

Describes simple integer data along with the standard subtest attributes.

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

-- subtestStringType --

Describes simple string data along with the standard subtest attributes.

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

-- subtestBaseType --

Describes complex data data along with the standard subtest attributes.

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

-- testedBoolType --

Describes simple boolean data along with the standard tested attributes.

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

-- testedIntType --

Describes simple integer data along with the standard tested attributes.

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

-- testedStringType --

Describes simple string data along with the standard tested attributes.

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

-- testedBaseType --

Describes complex data data along with the standard tested attributes.

Attributes:	(includes testedAttributes)
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 element in the defintion_object or definition_data section of a test. 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. The optional version attribute signifies which version of the variable was used during analysis. Different versions of a variable can be encounted if different values for the variable are used in different definition instances.

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

-- testedAttributes --

The following are the default attributes associated with every element in the tested_object section. The datatype attribute determines signifies whether the object was originally part of a pattern match or literal string. (the default datatype is 'string') The datatype attribute is optional in order to keep the XML clean and readable. The default value is used most of the time and putting datatype="literal" for each element would muddy up the XML.

Attributes:	[datatype]

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

Define acceptable check types. 'all' means to check that all matching object satisfy data requirements. 'at least one' means that at least one matching object satisfies the data requirements. 'none exists' means that no matching object exists that satisfy the data requirements. 'only one' means that 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 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.

-- component
-- binary
-- boolean
-- float
-- int
-- 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 platforms.

-- aix
-- apache
-- debian
-- freebsd
-- hp-ux
-- ios
-- macos
-- openbsd
-- oracle
-- os400
-- pix
-- redhat
-- solaris
-- suse
-- windows

instanceType values

Instance ids are simply positive integers.

objectDatatypes values

Define acceptable data types for an element in an tested_object section.

-- literal
-- pattern match

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

resultType values

Define acceptable result values.

-- -1
-- 0
-- 1

status values

The status of an OVAL definition.

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

statusType values

-- error
-- exists
-- does not exist

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}'
```

uknResultType values

Define acceptable result value for an unknown_test.

```
-- -1
```

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]*)'
```

versionType values

Version numbers are simply inetegers greater than or equal to 0.