- Open Vulnerability and Assessment Language - Element Dictionary

- Schema: Core Common
- Version: 5.2
- Release Date: 31 January 2007

The following is a description of the common types that are shared across the different schemas within Open Vulnerability and Assessment Language (OVAL). Each type is described in detail and should provide the information necessary to understand what each represents. This document is intended for developers and assumes some familiarity with XML. A high level description of the interaction between these type 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.

== GeneratorType ==

The GeneratorType complex type defines an element that is used to hold information about when a particular OVAL document was compiled, what version of the schema was used, waht 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 OVAL Schema. Individual organizations can place generator information that they feel are important and these will be skipped during the validation. All OVAL really cares about is that the stated generator information is there.

Child Elements	Type	MinOccurs	MaxOccurs
product_name	xsd:string	0	1
product_version	xsd:string	0	1
schema_version	xsd:decimal	1	1
timestamp	xsd:dateTime	1	1

== MessageType ==

The MessageType complex type defines the structure for which messages are relayed from the data collection engine. Each message is a text string that has an associated level attribute identifying the type of message being sent. These messages could be error messages, warning messages, debug messages, etc. How the messages are used by tools and whether or not they are displayed to the user is up to the specific implementation. Please refer to the description of the MessageLevelEnumeration for more information about each type of message.

Attributes:

- level oval:MessageLevelEnumeration (optional -- default='info')

Simple Content	xsd:string

-- CheckEnumeration --

The CheckEnumeration simple type defines acceptable check values, which determine how to evaluate multiple individual cases. When used to define the relationship between objects and states, each check value defines how many of the matching objects must satisfy the given state for the test to return true. When used to define the relationship between multiple values of entities, each check value defines how many values must be true for the entity to return true. When used to define the relationship between entities and multiple variable values, each check value defines how many variable values must be true for the entity to return true.

Value	Description
all	A value of 'all' means that a test returns true if a matching object exists and that all matching objects satisfy the data requirements for a test to evaluate to true.
at least one	A value of 'at least one' means that a test returns true if a matching object exists and at least one matching object must satisfies the data requirements for a test to evaluate to true.
none exist	A value of 'none exists' means that a test evaluates to true if no matching object exists that satisfy the data requirements.
only one	A value of 'only one' means that a test evaluates to true if a matching object exists and if one, and only one, matching object satisfies the data requirements.

Below are some tables that outline how each check attribute effects evaluation. The far left column identifies the check attribute in question. The middle column specifies the different combinations of individual results that the check attribute may bind together. (T=true, F=false, E=error, U=unknown, NE=not evaluated, NA=not applicable) For example, a 1+ under T means that one or more individual results are true, while a 0 under U means that zero individual results are unknown. The last column specifies what the final result would be according to each combination of individual results. Note that if the individual test is negated, then a true result is false and a false result is true, all other results stay as is.

	num of individual results						
check attr is	T	F	E	U	NE	NA	final result is
ALL	1+	0	0	0	0	0+	True
	0+	0+	0+	0+	0+	0+	False
	0+	0	1+	0+	0+	0+	Error
	0+	0	0	1+	0+	0+	Unknown
	0+	0	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

	num of individual results						
check attr is	T	F	E	U	NE	NA	final result is
	1+	0+	0+	0+	0+	0+	True
	0	0+	0	0	0	0+	False

AT LEAST ONE	0	0+	1+	0+	0+	0+	Error
	0	0+	0	1+	0+	0+	Unknown
	0	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

	num of individual results						
check attr is							final result is
	T	F	E	U	NE	NA	
ONLY ONE	1	0+	0	0	0	0+	True
	2+	0+	0+	0+	0+	0+	** False **
	0	0+	0	0	0	0+	** False **
	1-	0+	1+	0+	0+	0+	Error
	1-	0+	0	1+	0+	0+	Unknown
	1-	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

	num of individual results						
check attr is							final result is
	T	F	E	U	NE	NA	
NONE	0	0+	0	0	0	0+	True
	1+	0+	0+	0+	0+	0+	False
	0	0+	1+	0+	0+	0+	Error
	0	0+	0	1+	0+	0+	Unknown
	0	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

-- DatatypeEnumeration --

The DatatypeEnumeration simple type defines the legal datatypes that are used to describe the values of individual entities. 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'. Another example is applying the 'equal' operation to '1.0.0.0' and '1.0'. With datatype 'string' they are not equal, with datatype 'version' they are.

Value	Description
binary	The binary datatype is used to represent data that is in raw (non-printable) form. Values should be hex strings. Expected operations within OVAL for binary values are 'equals' and 'not equals'.
boolean	The boolean datatype represent standard boolean data, either true or false. Expected operations within OVAL for boolean values are 'equals' and 'not equals'.
evr_string	The evr_string datatype represents the epoch, version, and release fields as a single version string. It has the form "EPOCH:VERSION-RELEASE". Comparisons involving this datatype should follow the algorithm of librpm's rpmvercmp() function. Expected operations within OVAL for evr_string values are 'equals', 'not equals', 'greater than', 'greater than or equal', 'less

	than', and 'less than or equal'.
float	The float datatype describes standard float data. Expected operations within OVAL for float values are 'equals', 'not equals', 'greater than', 'greater than or equal', 'less than', and 'less than or equal'.
ios_version	The ios_version datatype describes Cisco IOS Train strings. These are in essence version strings for IOS. Please refer to Cisco's IOS Reference Guide for information on how to compare different Trains as they follow a very specific pattern. Expected operations within OVAL for ios_version values are 'equals', 'not equals', 'greater than', 'greater than or equal', 'less than', and 'less than or equal'.
int	The int datatype describes standard integer data. Expected operations within OVAL for int values are 'equals', 'not equals', 'greater than', 'greater than or equal', 'less than', 'less than or equal', 'bitwise and', and 'bitwise or'.
string	The string datatype describes standard string data. Expected operations within OVAL for string values are 'equals', 'not equals', 'pattern match'.
version	The version datatype represents a value that is a hierarchical list of integers separated by a single character delimiter. Expected operations within OVAL for version values are 'equals', 'not equals', 'greater than', 'greater than or equal', 'less than', and 'less than or equal'. 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.

	Version strings with a different number of components shall be padded with zeros to make them the same size. For example, if the version strings '1.2.3' and '6.7.8.9' are being compared, then the short one should be padded to become '1.2.3.0'.

-- FamilyEnumeration --

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

Value	Description
ios	
macos	
unix	
windows	

-- MessageLevelEnumeration --

The MessageLevelEnumeration simple type defines the different levels associated with a message. There is no specific criteria about which messages get assigned which level. This is completely arbitrary and up to the content producer to decide what is an error message and what is a debug message.

Value	Description
debug	Debug messages should only be displayed by a tool when run in some sort of verbose mode.
error	Error messages should be recorded when there was an error that did not allow the collection of specific data.
fatal	A fatal message should be recorded when an error causes the failure of more than just a single piece of data.
info	Info messages are used to pass useful information about the data collection to a user.
warning	A warning message reports something that might not correct but information was still collected.

-- OperationEnumeration --

The OperationEnumeration simple type defines acceptable operations. Each operation defines how to compare entities against their actual values.

Value	Description
equals	The 'equals' operation returns true if the actual value on the system is equal to the stated entity.
not equal	The 'not equal' operation returns true if the actual value on the system is not equal to the stated entity.

greater than	The 'greater than' operation returns true if the actual value on the system is greater than the stated entity.
less than	The 'less than' operation returns true if the actual value on the system is less than the stated entity.
greater than or equal	The 'greater than or equal' operation returns true if the actual value on the system is greater than or equal to the stated entity.
less than or equal	The 'less than or equal' operation returns true if the actual value on the system is less than or equal to the stated entity.
bitwise and	The 'bitwise and' operation is used to determine if a specific bit is set. It returns true if performing a BITWISE AND with the binary representation of the stated entity against the binary representation of the actual value on the system results in a binary value that is equal to the binary representation of the stated entity. For example, assuming a datatype of 'int', if the actual integer value of the setting on your machine is 6 (same as 0110 in binary), then performing a 'bitwise and' with the stated integer 4 (0100) returns 4 (0100). Since the result is the same as the state mask, then the test returns true. If the actual value on your machine is 1 (0001), then the 'bitwise and' with the stated integer 4 (0100) returns 0 (0000). Since the result is not the same as the stated mask, then the test fails.
bitwise or	The 'bitwise or' operation is used to determine if a specific bit is not set. It returns true if performing a BITWISE OR with the binary representation of the stated entity against the binary representation of the actual value on the system results in a binary value that is equal to the binary representation of the stated entity. For example, assuming a datatype of 'int', if the actual integer value of the setting on your machine is 6 (same as 0110 in binary), then performing a 'bitwise or' with the stated integer 14 (1110) returns 14 (1110). Since the result is the same as the state mask, then the test returns true. If the actual value on your machine is 1 (0001), then the 'bitwise or' with the stated integer 14 (1110) returns 15 (1111). Since the result is not the same as the stated mask, then the test fails.
pattern match	The 'pattern match' operation allows an item to be tested against a regular expression. Patterns must comply with POSIX std 1003.2-1992, Section 2.8 - 'Regular Expression Notation'. Patterns can use both Basic and Extended Regular Expression notation.

-- OperatorEnumeration --

The OperatorEnumeration simple type defines acceptable operators. Each operator defines how to evaluate multiple arguments.

Value	**Description**
AND	The AND operator produces a true result if every argument is true. If one or more arguments are false, the result of the AND is false. If one or more of the arguments are unknown, and if none of the arguments are false, then the AND operator produces a result of unknown.

OR	The OR operator produces a true result if one or more arguments is true. If every argument is false, the result of the OR is false. If one or more of the arguments are unknown, and if none of true arguments are true, then the OR operator produces a result of unknown.
XOR	XOR is defined to be true if an odd number of its arguments are true, and false otherwise. If any of the arguments are unknown, then the XOR operator produces a result of unknown.

Below are some tables that outline how each operator effects evaluation. The far left column identifies the operator in question. The middle column specifies the different combinations of individual results that the operator may bind together. (T=true, F=false, E=error, U=unknown, NE=not evaluated, NA=not applicable) For example, a 1+ under T means that one or more individual results are true, while a 0 under U means that zero individual results are unknown. The last column specifies what the final result would be according to each combination of individual results. Note that if the individual test is negated, then a true result is false and a false result is true, all other results stay as is.

operator is	num of individual results: T	F	E	U	NE	NA	final result is
AND	1+	0	0	0	0	0+	True
	0+	1+	0+	0+	0+	0+	False
	0+	0	1+	0+	0+	0+	Error
	0+	0	0	1+	0+	0+	Unknown
	0+	0	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

operator is	num of individual results: T	F	E	U	NE	NA	final result is
OR	1+	0+	0+	0+	0+	0+	True
	0	1+	0	0	0	0+	False
	0	0+	1+	0+	0+	0+	Error
	0	0+	0	1+	0+	0+	Unknown
	0	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

operator is	num of individual results: T	F	E	U	NE	NA	final result is
XOR	odd	0+	0	0	0	0+	True
	even	0+	0	0	0	0+	False
	0+	0+	1+	0+	0+	0+	Error
	0+	0+	0	1+	0+	0+	Unknown
	0+	0+	0	0	1+	0+	Not Evaluated
	0	0	0	0	0	1+	Not Applicable

-- DefinitionIDPattern --

Define the format for acceptable OVAL Definition ids. An urn format is used with the id starting with the word oval followed by a unique string, followed by the three letter code 'def', and ending with an integer.

oval:[A-Za-z0-9_\-\.]+:def:[1-9][0-9]*

-- ObjectIDPattern --

Define the format for acceptable OVAL Object ids. An urn format is used with the id starting with the word oval followed by a unique string, followed by the three letter code 'obj', and ending with an integer.

oval:[A-Za-z0-9_\-\.]+:obj:[1-9][0-9]*

-- StateIDPattern --

Define the format for acceptable OVAL State ids. An urn format is used with the id starting with the word oval followed by a unique string, followed by the three letter code 'ste', and ending with an integer.

oval:[A-Za-z0-9_\-\.]+:ste:[1-9][0-9]*

-- TestIDPattern --

Define the format for acceptable OVAL Test ids. An urn format is used with the id starting with the word oval followed by a unique string, followed by the three letter code 'tst', and ending with an integer.

oval:[A-Za-z0-9_\-\.]+:tst:[1-9][0-9]*

-- VariableIDPattern --

Define the format for acceptable OVAL Variable ids. An urn format is used with the id starting with the word oval followed by a unique string, followed by the three letter code 'var', and ending with an integer.

oval:[A-Za-z0-9_\-\.]+:var:[1-9][0-9]*

-- ItemIDPattern --

Define the format for acceptable OVAL Item ids. The format is an integer. An item id is used to identify the different items found in an OVAL System Characteristics file.

-- EmptyStringType --

The EmptyStringType simple type is a restriction of the built-in string simpleType. The only allowed string is the empty string with a length of zero. This type is used by certain elements to allow empty content when non-string data is accepted.

See the EntityIntType in the OVAL Definition Schema for an example of its use.