

- Open Vulnerability and Assessment Language - Element Dictionary

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

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. Some of the objects defined here are extended and enhanced by individual component 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 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>.

< oval_definitions >

The oval_definitions element is the root of an OVAL Definition Document. Its purpose is to bind together the major sections of a document - generator, definitions, tests, objects, states, and variables - which are the children of the root element.

Child Elements	Type	MinOccurs	MaxOccurs
generator	oval:GeneratorType	1	1
definitions	oval-def:DefinitionsType	0	1
tests	oval-def:TestsType	0	1
objects	oval-def:ObjectsType	0	1
states	oval-def:StatesType	0	1
variables	oval-def:VariablesType	0	1
ds:Signature	n/a	0	1

== DefinitionsType ==

The DefinitionsType complex type is a container for one or more definition elements. Each definition element describes a single OVAL Definition. Please refer to the description of the DefinitionType for more information about an individual definition.

Child Elements	Type	MinOccurs	MaxOccurs
definition	oval-def:DefinitionType	1	unbounded

== DefinitionType ==

The DefinitionType defines a single OVAL Definition. A definition is the key structure in OVAL. It is analogous to the 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. The DefinitionType contains a section for various metadata related elements that describe the definition. This includes a description, version, affected system types, and reference information. The notes section of a definition should be used to hold information that might be helpful to someone examining the technical aspects of the definition. For example, why certain tests have been included in the criteria, or maybe a link to where further information can be found. The DefinitionType also (unless the definition is deprecated) contains a criteria child element that 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. The form of an OVAL-ID must follow the specific format described by the definitionidPattern. The required version attribute holds the current version of the definition. Versions are integers, starting at 1 and incrementing every time a definition is modified. The required class attribute indicates the specific class to which the definition belongs. See the definition of classEnumeration for more information about the different valid classes. The optional deprecated attribute signifies that an id is no longer to be used or referenced but the information has been kept around for historic purposes.

Attributes:

-
- id oval:DefinitionIDPattern (required)
 - version xsd:integer (required)
 - class [oval-def:ClassEnumeration](#) (required)
 - deprecated xsd:boolean (optional -- default='false')

Child Elements	Type	MinOccurs	MaxOccurs
ds:Signature	n/a	0	1
metadata	oval-def:MetadataType	1	1
notes	oval-def:NotesType	0	1
criteria	oval-def:CriteriaType	0	1

== MetadataType ==

The MetadataType complex type contains all the metadata available to an OVAL Definition. This metadata is for informational purposes only and is not part of the criteria used to evaluate machine state. The required title child element holds a short string that is used to quickly identify the definition to a human user. The affected metadata item contains information about the system(s) for which the definition has been written. Remember that this is just metadata and not part of the criteria. Please refer to the AffectedType description for more information. The required 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.

Additional metadata is also allowed although it is not part of the official OVAL Schema. Individual organizations can place metadata items that they feel are important and these will be skipped during the validation. All OVAL really cares about is that the stated metadata items are there.

Child Elements	Type	MinOccurs	MaxOccurs
title	xsd:string	1	1
affected	oval-def:AffectedType	0	unbounded

reference	oval-def:ReferenceType	0	unbounded
description	xsd:string	1	1

== AffectedType ==

Each OVAL Definition is written to evaluate a certain type of system(s). The family, platform(s), and product(s) of this target are described by the AffectedType whose main purpose is to provide hints for tools using OVAL Definitions. For instance, to help a reporting tool only use Windows definitions, or to pre-select only Red Hat definitions to be evaluated. 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 AffectedType complex type details the specific system, 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 platform or product element can be thought of as definition applying to all platforms or products. 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. 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.

Please note that the AffectedType will change in future versions of OVAL in order to support the Common Platform Enumeration (CPE).

Attributes:

-
- family oval:FamilyEnumeration (required)

Child Elements	Type	MinOccurs	MaxOccurs
platform	xsd:string	0	unbounded
product	xsd:string	0	unbounded

== ReferenceType ==

The ReferenceType complex type 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 address the same issue being specified by the OVAL Definition.

The required source attribute specifies where the reference is coming from. In other words, it identifies the reference repository being used. The required ref_id attribute is the external id of the reference. The optional ref_url attribute is the URL to the reference.

Attributes:

-
- source xsd:string (required)
 - ref_id xsd:string (required)
 - ref_url xsd:anyURI (optional)

== NotesType ==

The NotesType complex type is a container for one or more note child elements. Each note contains some information about the definition or tests that it references. A note may record an unresolved question about the definition or test or present the reason as to why a particular approach was taken.

Child Elements	Type	MinOccurs	MaxOccurs
note	xsd:string	1	unbounded

== CriteriaType ==

The CriteriaType complex type describes the high level container for all the tests and represents the meat of the definition. Each criteria can contain other criteria elements in a recursive structure allowing complex logical trees to be constructed. Each referenced test is represented by a criterion element. Please refer to the description of the CriterionType for more information about and individual criterion element. The optional extend_definition element allows existing definitions to be included in the criteria. Refer to the description of the ExtendDefinitionType for more information.

The required operator attribute provides the logical operator that binds the different statements inside a criteria together. The optional negate attribute signifies that the result of the criteria as a whole should be negated during analysis. For example, consider a criteria that evaluates to TRUE if a certain software is installed. By negating this test, it now evaluates to TRUE if the software is NOT installed. The optional comment attribute provides a short description of the criteria.

Attributes:

- operator oval:OperatorEnumeration (optional -- default='AND')
- negate xsd:boolean (optional -- default='false')
- comment xsd:string (optional)

Child Elements	Type	MinOccurs	MaxOccurs
criteria	oval-def:CriteriaType		
criterion	oval-def:CriterionType		
extend_definition	oval-def:ExtendDefinitionType		

== CriterionType ==

The CriterionType complex type identifies a specific test to be included in the 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 evaluates to TRUE if a specific patch is installed. By negating this test, it now evaluates to TRUE if the patch is NOT installed. The optional comment attribute provides a short description of the specified test and should mirror the comment attribute of the actual test.

Attributes:

- test_ref oval:TestIDPattern (required)
- negate xsd:boolean (optional -- default='false')
- comment xsd:string (optional)

== ExtendDefinitionType ==

The ExtendDefinitionType complex type allows existing definitions to be extended by another definition. This works by

evaluating the extended definition and then using the result within the logical context of the extending definition.

The required `definition_ref` attribute is the actual id of the definition being extended. The optional `negate` attribute signifies that the result of an extended definition should be negated during analysis. For example, consider a definition that evaluates TRUE if a certain software is installed. By negating the definition, it now evaluates to TRUE if the software is NOT installed. The optional `comment` attribute provides a short description of the specified definition and should mirror the title metadata of the extended definition.

Attributes:

- `definition_ref` `oval:DefinitionIDPattern` (required)
- `negate` `xsd:boolean` (optional -- default='false')
- `comment` `xsd:string` (optional)

== TestType ==

The `TestType` complex type is a container for one or more test child elements. Each test element describes a single OVAL Test. Please refer to the description of the `TestType` for more information about an individual test.

Child Elements	Type	MinOccurs	MaxOccurs
oval-def:test	n/a	1	unbounded

< test >

The test element is an abstract element that is meant to be extended (via substitution groups) by the tests found in the component schemas. An actual test element is not valid. The use of this abstract class simplifies the OVAL schema by allowing individual tests to inherit the optional `notes` child element, and the `id` and `comment` attributes from the base `TestType`. Please refer to the description of the `TestType` complex type for more information.

== TestType ==

The base type of every test includes an optional `notes` element and five attributes. The `notes` section of a test should be used to hold information that might be helpful to someone examining the technical aspects of the test. For example, why certain values have been used by the test, or maybe a link to where further information can be found. Please refer to the description of the `NotesType` complex type for more information about the `notes` element.

The required `id` attribute uniquely identifies each test, and must conform to the format specified by the `testidPattern` simple type. The required `version` attribute holds the current version of the test. Versions are integers, starting at 1 and incrementing every time a test is modified. The required `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 valid check values are explained in the description of the `checkEnumeration` simple type. The required `comment` attribute provides a short description of the test. The optional `deprecated` attribute signifies that an id is no longer to be used or referenced but the information has been kept around for historic purposes.

Attributes:

- `id` `oval:TestIDPattern` (required)
- `version` `xsd:integer` (required)
- `check` `oval:CheckEnumeration` (required)

information that might be helpful to someone examining the technical aspects of the object. For example, why certain values have been used, or maybe a link to where further information can be found. Please refer to the description of the NotesType complex type for more information about the notes element.

The required id attribute uniquely identifies each object, and must conform to the format specified by the objectIdPattern simple type. The required version attribute holds the current version of the object element. Versions are integers, starting at 1 and incrementing every time an object is modified. The optional comment attribute provides a short description of the object. The optional deprecated attribute signifies that an id is no longer to be used or referenced but the information has been kept around for historic purposes.

Attributes:

- id oval:ObjectIDPattern (required)
- version xsd:integer (required)
- comment xsd:string (optional)
- deprecated xsd:boolean (optional -- default='false')

Child Elements	Type	MinOccurs	MaxOccurs
ds:Signature	n/a	0	1
notes	oval-def:NotesType	0	1

< set >

The set element enables complex objects to be described. It is a recursive element in that each set element can contain additional set elements as children. Each set element defines characteristics that produce a matching set of objects. The possible characteristics are an object reference and a collection of filters. The object_reference refers to an existing OVAL Object. The filter element provides a reference to an existing OVAL State. A filter is used to eliminate certain object from the set. Each filter is applied to each OVAL Object before the set_operator is applied. For example, if an object_reference points to an OVAL Object that is every file in a certain directory, a filter might be set up to limit the object set to only those files with a size less than 10 KB. If multiple filters are provided, then each filter is used separately against the defined object set. In other words, if an object matches any of the supplied filters, then it is thrown out of the set.

The required set_operator attribute defines how different child sets are combined to form the overall set of objects. For example, does one take the union of different sets or the intersection? For a description of the valid values please refer to the SetOperatorEnumeration simple type.

Attributes:

- set_operator [oval-def:SetOperatorEnumeration](#) (optional -- default='UNION')

Child Elements	Type	MinOccurs	MaxOccurs
object_reference	oval:ObjectIDPattern	1	2
filter	oval:StateIDPattern	0	unbounded

== StatesType ==

The StatesType complex type is a container for one or more state child elements. Each state provides details about specific characteristics that can be used during an evaluation of an object. Please refer to the description of the state element for

more information about an individual state.

Child Elements	Type	MinOccurs	MaxOccurs
oval-def:state	n/a	1	unbounded

< state >

The state element is an abstract element that is meant to be extended (via substitution groups) by the states found in the component schemas. An actual state element is not valid. The use of this abstract class simplifies the OVAL schema by allowing individual states to inherit the optional notes child element, and the id and operator attributes from the base StateType. Please refer to the description of the StateType complex type for more information.

== StateType ==

The base type of every state includes an optional notes element and two attributes. The notes section of a state should be used to hold information that might be helpful to someone examining the technical aspects of the state. For example, why certain values have been used by the state, or maybe a link to where further information can be found. Please refer to the description of the NotesType complex type for more information about the notes element.

The required id attribute uniquely identifies each state, and must conform to the format specified by the stateidPattern simple type. The required version attribute holds the current version of the state. Versions are integers, starting at 1 and incrementing every time a state is modified. The required operator attribute provides the logical operator that binds the different characteristics inside a state together. The optional comment attribute provides a short description of the state. The optional deprecated attribute signifies that an id is no longer to be used or referenced but the information has been kept around for historic purposes.

When evaluating a particular state against an object, one should evaluate each individual entity separately. The individual results are then combined by the operator to produce an overall result. This process holds true even when there are multiple instances of the same entity. Evaluate each instance separately, taking the entity check attribute into account, and then combine everything using the operator.

Attributes:

- id oval:StateIDPattern (required)
- version xsd:integer (required)
- operator oval:OperatorEnumeration (optional -- default='AND')
- comment xsd:string (optional)
- deprecated xsd:boolean (optional -- default='false')

Child Elements	Type	MinOccurs	MaxOccurs
ds:Signature	n/a	0	1
notes	oval-def:NotesType	0	1

== VariablesType ==

The VariablesType complex type is a container for one or more variable child elements. Each variable element is a way to define one or more values to be obtained at the time a definition is evaluated.

Child Elements	Type	MinOccurs	MaxOccurs
oval-def:variable	n/a	1	unbounded

< variable >

The variable element is an abstract element that is meant to be extended (via substitution groups) by the different types of variables. An actual variable element is not valid. The different variable types describe different sources for obtaining a value(s) for the variable. There are currently three types of variables; local, external, and constant. Please refer to the description of each one for more specific information. The value(s) of a variable is treated as if it were inserted where referenced. One of the main benefits of variables is that they allow tests to evaluate user-defined policy. For example, an OVAL Test might check to see if a password is at least a certain number of characters long, 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 that defines the length.

If a variable defines an array of values, any entity that references the variable will evaluate to true depending on the value of the var_check attribute. For example, if an entity 'size' with an operation of 'less than' references a variable that returns five different integers, and the var_check attribute has a value of 'all', then the 'size' entity returns true only if the actual size is less than each of the five integers defined by the variable. If a variable does not return any value, then an error should be thrown during OVAL analysis.

== VariableType ==

The VariableType complex type defines attributes associated with each OVAL Variable. The required id attribute uniquely identifies each variable, and must conform to the format specified by the varidPattern simple type. The required version attribute holds the current version of the variable. Versions are integers, starting at 1 and incrementing every time a variable is modified. The required datatype attribute specifies the type of value being defined. The required comment attribute provides a short description of the variable. The optional deprecated attribute signifies that an id is no longer to be used or referenced but the information has been kept around for historic purposes.

Attributes:

-
- id oval:VariableIDPattern (required)
 - version xsd:integer (required)
 - datatype oval:DatatypeEnumeration (required)
 - comment xsd:string (required)
 - deprecated xsd:boolean (optional -- default='false')

Child Elements	Type	MinOccurs	MaxOccurs
ds:Signature	n/a	0	1

< external_variable >

The external_variable element extends the VariableType and defines a variable with some external source. The actual value(s) for the variable is not provided within the OVAL file, but rather it is retrieved during the evaluation of the OVAL Definition from an external source. An unbounded set of possible_value and possible_restriction child elements can be specified that together specify the list of all possible values that an external source is allowed to supply for the external variable. In other words, the value assigned by an external source must match one of the possible_value or possible_restriction elements specified. Each possible_value element contains a single value that could be assigned to the given external_variable while each possible_restriction element outlines a range of possible values. Note that it is not necessary to declare a variable's possible values, but the option is available if desired. If no possible child elements are specified, then the valid values are only bound to the specified datatype of the external variable. Please refer to the

description of the PossibleValueType and PossibleRestrictionType complex types for more information.

== PossibleValueType ==

The PossibleValueType complex type is used to outline a single expected value of an external variable. The required hint attribute gives a short description of what the value means or represents.

Attributes:

- hint xsd:string (required)

Simple Content	xsd:anySimpleType
-----------------------	-------------------

== PossibleRestrictionType ==

The PossibleRestrictionType complex type outlines a range of possible expected value of an external variable. Each possible_restriction element contains an unbounded list of child restriction elements that each specify a range that an actual value may fall in. For example, a restriction element may specify that a value must be less than 10. When multiple restriction elements are present, a valid possible value would have to meet every restriction. One can think of the possible_value and possible_restriction elements as an OR'd list of possible values, and the restriction elements as an AND'd list of value descriptions. Please refer to the description of the RestrictionType complex type for more information. The required hint attribute gives a short description of what the value means or represents.

Attributes:

- hint xsd:string (required)

Child Elements	Type	MinOccurs	MaxOccurs
restriction	oval-def:RestrictionType	1	unbounded

== RestrictionType ==

The RestrictionType complex type outlines a restriction that is placed on expected values for an external variable. For example, a possible value may be restricted to a integer less than 10. Please refer to the operationEnumeration simple type for a description of the valid operations. The required hint attribute gives a short description of what the value means or represents.

Attributes:

- operation oval:OperationEnumeration (required)

Simple Content	xsd:anySimpleType
-----------------------	-------------------

< constant_variable >

The constant_variable element extends the VariableType and defines a variable with a constant value(s). Each constant_variable defines either a single value or an array of values to be used throughout the evaluation of the OVAL Definition File in which it has been defined. Constant variables can not be over-ridden by an external source. The actual value of a constant variable is defined by the required value child element. An array of values can be specified by

including multiple instances of the value element. Please refer to the description of the ValueType complex type for more information.

Child Elements	Type	MinOccurs	MaxOccurs
value	oval-def:ValueType	1	unbounded

== ValueType ==

The ValueType complex type holds the actual value of the variable when dealing with a constant variable. This value should be used by all tests that reference this variable. The value can not be over-ridden by an external source.

Attributes:

Simple Content	xsd:anySimpleType
-----------------------	-------------------

< local_variable >

The local_variable element extends the VariableType and defines a variable with some local source. The actual value(s) for the variable is not provided in the OVAL Definition document but rather it is retrieved during the evaluation of the OVAL Definition. A value can be as simple as a literal string or as complex as multiple registry keys concatenated together. Each local variable is defined by either a single component or a complex function. Please refer to the description of the ComponentGroup for more information.

Child Elements	Type	MinOccurs	MaxOccurs
oval-def:ComponentGroup	n/a	1	1

-- ComponentGroup --

Any value that is pulled directly off the local system is defined by the basic component element. For example, the name of a user or the value of a registry key. Please refer to the definition of the ObjectComponentType for more information. A value can also be obtained from another variable. The variable element identifies a variable id to pull a value(s) from. Please refer to the definition of the VariableComponentType for more information. Literal values can also be specified.

Child Elements	Type	MinOccurs	MaxOccurs
object_component	oval-def:ObjectComponentType		
variable_component	oval-def:VariableComponentType		
literal_component	xsd:anySimpleType		
oval-def:FunctionGroup	n/a		

== ObjectComponentType ==

The ObjectComponentType complex type defines a specific value on the local system to obtain. The required obj_id provides a reference to an existing OVAL Object declaration. This object defines the object to examine and eventually pull the value from. The required item_field defines which piece of data to retrieve from the object referenced by the obj_id. For example, if the obj_id references a file, the item_field may define the version as the piece of information to use

as the value of the variable. The data to retrieve can be found in the OVAL System Characteristics file under the items associated with the object referenced by obj_id.

Attributes:

-
- object_ref oval:ObjectIDPattern (required)
 - item_field xsd:string (required)

== VariableComponentType ==

The VariableComponentType complex type defines a specific value obtained by looking at the value of another OVAL Variable. The required var_ref attribute provides a reference to the variable. One must make sure that the variable reference does not point to the parent variable that uses this component to avoid a race condition.

Attributes:

-
- var_ref oval:VariableIDPattern (required)

-- FunctionGroup --

Complex functions have been defined that help determine how to manipulated specific values. These functions can be nested together to form complex statements. Each function is designed to work on a specific type of data. If the data being worked on is not of the correct type, a cast should be attempted before throwing an error. For example, if a concat function includes a registry component that returns an integer, then the integer should be cast as a string in order to work with the concat function. Note that if the operation being applied to the variable by the calling entity is "pattern match", then all the functions are performed before the regular expression is evaluated. In short, the variable would produce a value as normal and then any pattern match operation would be performed. Please refer to the description of a specific function for more details about it.

Child Elements	Type	MinOccurs	MaxOccurs
begin	oval-def:BeginFunctionType		
concat	oval-def:ConcatFunctionType		
end	oval-def:EndFunctionType		
escape_regex	oval-def:EscapeRegexFunctionType		
split	oval-def:SplitFunctionType		
substring	oval-def:SubstringFunctionType		

== BeginFunctionType ==

The begin function takes a single string component and defines a character (or string) that the component string should start with. The character attribute defines the specific character (or string). The character (or string) is only added to the component string if the component string doesn't already start with the specified character (or string).

Attributes:

-
- character xsd:string (required)

Child Elements	Type	MinOccurs	MaxOccurs

oval-def:ComponentGroup	n/a		
---	-----	--	--

== ConcatFunctionType ==

The concat function takes two or more components and concatenates them together to form a single string. The first component makes up the beginning of the resulting string and any following components are added to the end it. If one of the components returns multiple values then the concat function would be performed multiple times and the end result would be an array of values for the local variable. For example assume a local variable has two sub-components: a basic component element returns the values "abc" and "def", and a literal component element that has a value of "xyz". The local_variable element would be evaluated to have two values, "abcxyz" and "defxyz". If one of the components does not exist, then the result of the concat operation should be does not exist.

Child Elements	Type	MinOccurs	MaxOccurs
oval-def:ComponentGroup	n/a		

== EndFunctionType ==

The end function takes a single string component and defines a character (or string) that the component string should end with. The character attribute defines the specific character (or string). The character (or string) is only added to the component string if the component string doesn't already end with the specified character (or string).

Attributes:

- character xsd:string (required)

Child Elements	Type	MinOccurs	MaxOccurs
oval-def:ComponentGroup	n/a		

== EscapeRegexFunctionType ==

The escape regex function takes a single string component and escapes all the regular expression characters. The purpose for this is that many times, a component used in pattern match needs to be treated a literal string and not regular expression. For example assume a basic component element that pulls a file path out of the Windows registry. This path is a string that might contain regular expression characters but these characters are not intended to be such, so they need to be escaped. This function allows a definition writer to mark which components are in regular expression format and which aren't.

Child Elements	Type	MinOccurs	MaxOccurs
oval-def:ComponentGroup	n/a		

== SplitFunctionType ==

The split function takes a single string component and turns it into multiple values based on a delimiter string. For example assume a basic component element that returns the value "a-b-c-d" with the delimiter set to "-". The local_variable element would be evaluated to have four values "a", "b", "c", and "d". If the string component used by the split function returns multiple values, then the split is performed multiple times.

Attributes:

- delimiter xsd:string (required)

Child Elements	Type	MinOccurs	MaxOccurs
oval-def:ComponentGroup	n/a		

== SubstringFunctionType ==

The substring function takes a single string component and produces a single value that contains a portion of the original string. The `substring_start` attribute defines the starting position in the original string. Note, to include the first character of the string, the start position would be 1. Also note that a value less than one also means starting at the first character of the string. The `substring_length` attribute defines how many character after and including the starting character to include. Note that a `substring_length` value greater than the actual length of the string or a negative value means to include all the characters after the starting character. For example assume a basic component element that returns the value "abcdefg" with a `substring_start` value of 3 and a `substring_length` value of 2. The `local_variable` element would be evaluate to have a single value of "cd". If the string component used by the substring function returns multiple values, then the substring operation is performed multiple times and results in multiple values for the component.

Attributes:

- `substring_start` xsd:int (required)
- `substring_length` xsd:int (required)

Child Elements	Type	MinOccurs	MaxOccurs
oval-def:ComponentGroup	n/a		

-- ClassEnumeration --

The ClassEnumeration simple type defines the different classes of definitions. These classes are used to group definitions by the type of system state they are describing. For example, this allows users to find all the vulnerability definitions.

Value	Description
compliance	A compliance definition describes the state of a machine as it complies with a specific policy.
inventory	An inventory definition describes whether a specific piece of software is installed on the system.
miscellaneous	The 'miscellaneous' class is used to identify definitions that do not fall into any of the other defined classes.
patch	A patch definition details the machine state of whether a patch executable should be installed.
vulnerability	A vulnerability definition describes the conditions under which a machine is vulnerable.

-- SetOperatorEnumeration --

The SetOperatorEnumeration simple type defines acceptable set operations. Set operations are used to take multiple different sets of objects within OVAL and merge them into a single set. The different operators that guide this merge are

defined below. For each operator, if only a single object has been supplied, then the resulting set is simply that complete object.

Value	Description
COMPLEMENT	The complement operator is defined in OVAL as a relative complement. The resulting set contains everything that belongs to the first declared set that is not part of the second declared set. If A and B are sets (with A being the first declared set), then the relative complement is the set of elements in A, but not in B.
INTERSECTION	The intersection of two sets in OVAL results in a set that contains everything that belongs both sets in the collection, but nothing else. If A and B are sets, then the intersection of A and B contains all the elements of A that also belong to B, but no other elements.
UNION	The union of two sets in OVAL results in a set that contains everything that belongs to either of the original sets. If A and B are sets, then the union of A and B contains all the elements of A and all elements of B, with the duplicates removed.

== EntityBaseType ==

The EntityBaseType complex type is an abstract type that defines the default attributes associated with every entity. Entities can be found in both OVAL Objects and OVAL States and represent the individual properties associated with items found on a system. An example of a single entity would be the path of a file. Another example would be the version of the file.

The optional datatype attribute specifies how the given operation should be applied to the data. (the default datatype is 'string') An example is with the statement 'is 123 less than 98'. If the data is treated as integers the answer is no, but if the data is treated as strings, then the answer is yes. Specifying a datatype details how the less than operation should be performed. Another way of thinking of things is that the datatype attribute specifies how the data should be cast before performing the operation. In the previous example, if the datatype is set to int, then '123' and '98' should be cast as integers. If a cast can not be made, (trying to cast 'abc' to an integer) then an error should be thrown. 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.

The optional operation determines how the individual entities should be evaluated. (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 entity to a variable element. When supplied, the value(s) associated with the OVAL Variable should be used as the value(s) of the entity. If there is an error computing the value of the variable, then that error should be passed up to the entity referencing it. If the variable being referenced does not have a value (for example, if the variable pertains to the size of a file, but the file does not exist) then one of two results are possible. If the entity is part of an object declaration, then the object is considered to not exist. If the entity is part of a state declaration, then the state comparison should result in an error.

Attributes:

-
- datatype oval:DatatypeEnumeration (optional -- default='string')
 - operation oval:OperationEnumeration (optional -- default='equals')
 - var_ref oval:VariableIDPattern (optional)

Simple Content	xsd:anySimpleType
-----------------------	-------------------

== EntityObjectType ==

The EntityObjectType complex type is an abstract type that extends the EntityBaseType and is used by the entities within an OVAL Objects.

If the entity uses a `var_ref` and the associated variable defines more than one value, the optional `var_check` attribute defines how the data collection should proceed. For example, if an object entity 'filename' with an operation of 'does not equal' references a variable that returns five different values, and the `var_check` attribute has a value of 'all', then an actual file on the system matches only if the actual filename does not equal any of the variable values. If a variable does not return any value, then an error should be thrown during OVAL analysis.

Attributes:

-
- `var_check` `oval:CheckEnumeration` (optional -- default='all')

Simple Content	oval-def:EntityBaseType
-----------------------	-------------------------

== EntityObjectAnyType ==

The EntityObjectAnyType type is extended by the entities of an individual OVAL Object. This type provides uniformity to each object entity by including the attributes found in the EntityObjectType. This specific type describes any simple data.

Attributes:

Simple Content	oval-def:EntityObjectType
-----------------------	---------------------------

== EntityObjectBinaryType ==

The EntityBinaryType type is extended by the entities of an individual OVAL Object. This type provides uniformity to each object entity by including the attributes found in the EntityObjectType. This specific type describes simple binary data. The empty string is also allowed when using a variable reference with an element.

== EntityObjectBoolType ==

The EntityBoolType type is extended by the entities of an individual OVAL Object. This type provides uniformity to each object entity by including the attributes found in the EntityObjectType. This specific type describes simple boolean data. The empty string is also allowed when using a variable reference with an element.

== EntityObjectFloatType ==

The EntityObjectFloatType type is extended by the entities of an individual OVAL Object. This type provides uniformity to each object entity by including the attributes found in the EntityObjectType. This specific type describes simple float data. The empty string is also allowed when using a variable reference with an element.

== EntityObjectType ==

The EntityIntType type is extended by the entities of an individual OVAL Object. This type provides uniformity to each object entity by including the attributes found in the EntityObjectBaseType. This specific type describes simple integer data. The empty string is also allowed when using a variable reference with an element.

== EntityObjectStringType ==

The EntityStringType type is extended by the entities of an individual OVAL Object. This type provides uniformity to each object entity by including the attributes found in the EntityObjectBaseType. This specific type describes simple string data.

== EntityStateBaseType ==

The EntityStateBaseType complex type is an abstract type that extends the EntityBaseType and is used by the entities withing an OVAL State.

The optional entity_check attribute specifies how to handle entities with multiple instances in the system characteristics file. For example, if an OVAL Object has multiple values associated with it and the OVAL State defines the value entity as 'less than 3', the entity_check attribute determines if all values must be less than 3, or at least one value must be less than 3, etc.

If the state entity uses a var_ref and the associated variable defines more than one value, the optional var_check attribute defines how the evaluation should proceed. For example, if an entity 'size' with an operation of 'less than' references a variable that returns five different integers, and the var_check attribute has a value of 'all', then the 'size' entity returns true only if the actual size is less than each of the five integers defined by the variable. If a variable does not return any value, then an error should be thrown during OVAL analysis.

Attributes:

-
- entity_check oval:CheckEnumeration (optional -- default='all')
 - var_check oval:CheckEnumeration (optional -- default='all')

Simple Content	oval-def:EntityBaseType
-----------------------	-------------------------

== EntityStateAnyType ==

The EntityStateAnyType type is extended by the entities of an individual OVAL State. This type provides uniformity to each state entity by including the attributes found in the EntityStateBaseType. This specific type describes any simple data.

Attributes:

Simple Content	oval-def:EntityStateBaseType
-----------------------	------------------------------

== EntityStateBinaryType ==

The EntityStateBinaryType type is extended by the entities of an individual OVAL State. This type provides uniformity to

each state entity by including the attributes found in the EntityStateBaseType. This specific type describes simple binary data. The empty string is also allowed when using a variable reference with an element.

== EntityStateBoolType ==

The EntityStateBoolType type is extended by the entities of an individual OVAL State. This type provides uniformity to each state entity by including the attributes found in the EntityStateBaseType. This specific type describes simple boolean data. The empty string is also allowed when using a variable reference with an element.

== EntityStateFloatType ==

The EntityStateFloatType type is extended by the entities of an individual OVAL State. This type provides uniformity to each state entity by including the attributes found in the EntityStateBaseType. This specific type describes simple float data. The empty string is also allowed when using a variable reference with an element.

== EntityStateIntType ==

The EntityStateIntType type is extended by the entities of an individual OVAL State. This type provides uniformity to each state entity by including the attributes found in the EntityStateBaseType. This specific type describes simple integer data. The empty string is also allowed when using a variable reference with an element.

== EntityStateStringType ==

The EntityStateStringType type is extended by the entities of an individual OVAL State. This type provides uniformity to each state entity by including the attributes found in the EntityStateBaseType. This specific type describes simple string data.