Security Automation and Continuous Monitoring M. Cokus
Internet-Draft D. Haynes
Intended status: Informational D. Rothenberg
Expires: March 11, 2017 The MITRE Corporation
J. Gonzalez
Department of Homeland Security
September 7, 2016

OVAL(R) Definitions Model
draft-haynes-sacm-oval-definitions-model-01

Abstract

This document specifies Version 5.11.1 of the OVAL Definitions Model which defines an extensible framework for making assertions about a system that are based upon a collection of logical statements. Each logical statement defines a specific machine state by identifying the data set on the system to examine and describing the expected state of that system data.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 11, 2017.

Table of Contents

1. Introduction

The Open Vulnerability and Assessment Language (OVAL) [OVAL-WEBSITE] is an international, information security community effort to standardize how to assess and report upon the machine state of systems. For over ten years, OVAL has been developed in collaboration with any and all interested parties to promote open and publicly available security content and to standardize the representation of this information across the entire spectrum of security tools and services.

OVAL provides an established framework for making assertions about a system's state by standardizing the three main steps of the assessment process: representing the current machine state; analyzing the system for the presence of the specified machine state; and representing the results of the assessment which facilitates collaboration and information sharing among the information security community and interoperability among tools.

This draft is part of the OVAL contribution to the IETF SACM WG that standardizes the representation used to analyze a system for the presence of a specific machine state. It is intended to serve as a starting point for the endpoint posture assessment data modeling needs of SACM specifically Collection Guidance and Evaluation Guidance.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. oval_definitions

The oval_definitions type defines the base structure in the OVAL Definitions Model for representing a collection of OVAL Definitions. This container type adds metadata about the origin of the content and allows for a signature.

Property	Type	Count	Description
generator	oval:GeneratorType	1	Provides information regarding the origin of the OVAL Content. The timestamp property of the generator MUST represent the time at which the oval_definitions was created.
definitions	DefinitionsType	0..1	Container for OVAL Definitions.

tests	TestsType	0..1	Container for OVAL Tests.
objects	ObjectsType	0..1	Container for OVAL Objects.
states	StatesType	0..1	Container for OVAL Tests.
variables	VariablesType	0..1	Container for OVAL Variables.
signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the content.

Table 1: oval_definitions Construct

3. DefinitionsType

The DefinitionsType provides a container for one or more OVAL Definitions.

Property	Type	Count	Description
definition	DefinitionType	1..*	One or more OVAL Definitions.

Table 2: DefinitionsType Construct

4. DefinitionType

The DefinitionType defines a single OVAL Definition. An OVAL Definition is the key structure in the OVAL Definition Model. It is a collection of logical statements that combine to make an overall assertion about a system state and metadata about the assertion.

Property	Type	Count	Description
id	oval:DefinitionIDPattern	1	The globally unique identifier of the OVAL Definition.
version	unsigned integer	1	The version of the OVAL Definition.
class	oval:ClassEnumeration	1	The class of the OVAL Definition.
deprecated	boolean	0..1	Whether or not the OVAL Definition has been deprecated. Default Value: 'false'.
metadata	MetadataType	1	Container for metadata

| | | | | associated with the OVAL Definition. Metadata is informational only and does not impact the evaluation of the |

				OVAL Definition.
notes	NotesType	0..1	A container for individual notes that describe some aspect of the OVAL Definition.
criteria	CriteriaType	0..1	A container for the logical criteria that is defined by the OVAL Definition. All non-deprecated OVAL Definitions MUST contain at least one criteria to express the logical assertion being made by the OVAL Definition.
signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the content.

Table 3: DefinitionType Construct

5. MetadataType

The MetadataType is a container for additional metadata that describes an OVAL Definition.

Property	Type	Count	Description
title	string	1	A short text title for the OVAL Definition.
affected	AffectedType	0..*	A container for the list of affected platforms by a given

				OVAL Definition.
reference	ReferenceType	0..*	References allow pointers to external information about an OVAL Definition.
description	string	1	A detailed text description of the OVAL Definition.
extension_point	Any	0..*	An extension point that allows for the inclusion of any additional metadata associated with the OVAL Definition.

Table 4: MetadataType Construct

The extension_point property is not considered a part of the OVAL Language proper, but rather, an extension point that allows organizations to expand the OVAL Language to better suit their needs.

6. AffectedType

The AffectedType is a container type for the list of affected platforms and products. Note that the absence of a platform or product implies that the OVAL Definition applies to all platforms or products.

Property	Type	Count	Description
family	oval:FamilyEnumeration	1	The high-level classification of the system type.
platform	string	0..*	The name identifying a specific software platform. Convention is not to spell out the names.
product	string	0..*	The name identifying a specific software product. Convention is to spell out the names.

Table 5: AffectedType Construct

7. ReferenceType

The ReferenceType is a pointer to an external reference that supports or adds more information to an OVAL Definition.

Property	Type	Count	Description
source	string	1	The source of the reference.
ref_id	string	1	The identifier for the reference.

ref_url	URI	0..1	The URL for the reference.

Table 6: ReferenceType Construct

8. NotesType

The NotesType is a container for one or more notes, providing additional information, such as unresolved questions, reasons for specific implementation, or other documentation.

Property	Type	Count	Description
note	string	1..*	One or more text notes.

Table 7: NotesType Construct

9. CriteriaType

The CriteriaType defines the structure of a logical statement that combines other logical statements. This construct is used to combine references to OVAL Tests, OVAL Definitions, and other CriteriaTypes into one logical statement.

Property	Type	Count	Description
operator	oval:OperatorEnumeration	0..1	The logical operator that is used to combine the individual results of the logical statements defined by the criteria, criterion, and extend_definition properties. Default Value: 'AND'.
negate	boolean	0..1	Specifies whether or not the evaluation result of the CriteriaType should be negated. Default Value: 'false'.
comment	oval:NonEmptyStringType	0..1	A short description of the criteria.

criteria	CriteriaType	0..*	A collection of logical statements that

				will be combined according to the operator property. At least one criteria, criterion, or extend_definition MUST be present.
criterion	CriterionType	0..*		A logical statement that references an OVAL Test and will be combined according to the operator property. At least one criteria, criterion, or extend_definition MUST be present.
extend_definition	ExtendDefinitionType	0..*		A logical statement that references an OVAL Definition and will be combined according to the operator property. At least one criteria, criterion, or extend_definition MUST be present.
applicability_check	boolean	0..1		A boolean flag that when 'true' indicates that the criteria is being used to determine whether the OVAL Definition applies to a given system. No additional meaning is assumed when 'false'.

Table 8: CriteriaType Construct

10. CriterionType

The CriterionType is a logical statement that references an OVAL Test.

Property	Type	Count	Description
test_ref	oval:TestIDPattern	1	The globally unique

				identifier of an OVAL Test contained in the OVAL Definitions.
	negate	boolean	0..1	Specifies whether or not the evaluation result of the OVAL Test, referenced by the test_ref property, should be negated.

				Default Value: 'false'.
	comment	oval:NonEmptyStringType	0..1	A short description of the criterion.
	applicability_check	boolean	0..1	A boolean flag that when 'true' indicates that the criterion is being used to determine whether the OVAL Definition applies to a given system. No additional meaning is assumed when 'false'.

Table 9: CriterionType Construct

11. ExtendDefinitionType

The ExtendDefinitionType is a logical statement that references another OVAL Definition.

Property	Type	Count	Description
definition_ref	oval:DefinitionIDPattern	1	The globally unique identifier of an OVAL Definition contained in the OVAL Definitions.

negate	boolean	0..1	Specifies whether or not the evaluation result of the OVAL Definition, referenced by the definition_ref property, should be negated. Default Value: 'false'.
comment	oval:NonEmptyStringType	0..1	A short description of the extended OVAL Definition.
applicability_check	boolean	0..1	A boolean flag that when 'true' indicates that the ExtendDefinition is being used to determine whether the OVAL Definition applies to a given system. No additional meaning is assumed when 'false'.

Table 10: ExtendDefinitionType Construct

12. TestsType

The TestsType provides a container for one or more OVAL Tests.

Property	Type	Count	Description
test	TestType	1..*	One or more OVAL Tests.

Table 11: TestsType Construct

13. TestType

The TestType is an abstract OVAL Test that defines the common properties associated with all OVAL Tests. The TestType provides an extension point for concrete OVAL Tests, which define platform-specific capabilities in OVAL Component Models. An OVAL Test defines the relationship between an OVAL Object and zero or more OVAL States, specifying exactly how many OVAL Items must exist on the system and how many of those OVAL Items must satisfy the set of referenced OVAL States.

Property	Type	Count	Description

id	oval:TestIDPattern	1	The globally unique identifier of an OVAL Test.
version	unsigned int	1	The version of the unique OVAL Test.
check_existence	oval:ExistenceEnumeration	0..1	Specifies how many OVAL Items must exist, on the system, in order for the OVAL Test to evaluate to 'true'. Default Value: 'at_least_one_exists'.
check	oval:CheckEnumeration	1	Specifies how many of the collected OVAL Items must satisfy the requirements specified by the OVAL State(s) in order for the OVAL Test to evaluate to 'true'.
state_operator	oval:OperatorEnumeration	0..1	Specifies how to logically combine the OVAL States referenced in the OVAL Test. Default Value: 'AND'.
comment	oval:NonEmptyStringType	1	A short description of the OVAL Test. This value SHOULD describe the intent of the OVAL Test including the system information that is examined and the expected state of that information.
deprecated	boolean	0..1	Whether or not the OVAL Test has been deprecated. A deprecated OVAL Test is one that should no longer be referenced by new OVAL Content. Default Value: 'false'.
notes	NotesType	0..1	A container for individual notes that describe some aspect of the OVAL Test.
signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the content.

Table 12: TestType Construct

14. ObjectRefType

The ObjectRefType provides to an existing OVAL Object.

Property	Type	Count	Description
object_ref	oval:ObjectIDPattern	1	A reference to an existing OVAL Object.

Table 13: ObjectRefType Construct

15. StateRefType

The StateRefType provides to an existing OVAL State.

Property	Type	Count	Description
state_ref	oval:StateIDPattern	1	A reference to an existing OVAL State.

Table 14: StateRefType Construct

16. ObjectsType

The ObjectsType provides a container for one or more OVAL Objects.

Property	Type	Count	Description
object	ObjectType	1..*	A collection of OVAL Objects.

Table 15: ObjectsType Construct

17. ObjectType

The ObjectType is an abstract OVAL Object that defines the common properties associated with all OVAL Objects. The ObjectType provides an extension point for normal or "concrete" OVAL Objects, which define platform-specific capabilities, in the OVAL Component Models. A concrete OVAL Object MUST define sufficient entities to allow a user to identify a unique an item to be collected.

A concrete OVAL Object may define a set of 0 or more OVAL Behaviors. OVAL Behaviors define an action that can further specify the set of OVAL Items that match an OVAL Object. OVAL Behaviors may depend on other OVAL Behaviors or may be independent of other OVAL Behaviors. In addition, OVAL Behaviors are specific to OVAL Objects and are defined in the OVAL Component Models.

Property	Type	Count	Description
id	oval:ObjectIDPattern	1	The globally unique identifier of an OVAL Object contained in the OVAL Definitions.

version	unsigned int	1	The version of the globally unique OVAL Object referenced by the id property.
comment	oval:NonEmptyStringType	1	A short description of the OVAL Object.
deprecated	boolean	0..1	Whether or not the OVAL Object has been deprecated. Default Value: 'false'.
notes	NotesType	0..1	A container for individual notes that describe some aspect of the OVAL Object.
signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the content.

Table 16: ObjectType Construct

18. set

The set construct enables the expression of complex OVAL Objects that are the result of logically combining and filtering the OVAL Items that are identified by one or more other OVAL Objects. A set can consist of either one or two nested sets or one or two references to other OVAL Objects and a collection of OVAL Filters.

Property	Type	Count	Description
set_operator	SetOperatorEnumeration	0..1	Specifies the set operation to use when combining subsets. Default Value: 'UNION'.
set	set	0..2	Allowed nested sets.
object_reference	oval:ObjectIDPattern	0..2	A reference to an OVAL Object based upon its ID. An object_reference indicates that any OVAL Items identified by the referenced OVAL Object are included in the set. The referenced OVAL Object MUST be contained within the current instance of the OVAL Definitions Model and MUST be of the same type as the OVAL Object that is referencing it.
filter	filter	0..n	Defines one or more filters to apply to combined data.

Table 17: set Construct

19. filter

The filter construct allows the explicit inclusion or exclusion of OVAL Items from a collection of OVAL Items based upon one an OVAL State.

Property	Type	Count	Description
action	FilterActionEnumeration	0..1	Defines the type of

				filter. Default Value: 'exclude'.
value	oval:StateIDPattern	1		A reference to an OVAL State that defines how the data should be filtered. The referenced OVAL State MUST be contained within the current instance of the OVAL Definitions Model and MUST be of the same type as the OVAL Object that is referencing it.

Table 18: filter Construct

20. StatesType

The StatesType provides a container for one or more OVAL States.

Property	Type	Count	Description
state	StateType	1..*	A collection of OVAL States.

Table 19: StatesType Construct

21. StateType

The StateType is an abstract OVAL State that defines the common properties associated with all OVAL States. The StateType provides an extension point for concrete OVAL States, which define platform-specific capabilities in the OVAL Component Models. The StateType is

extended by concrete OVAL States in order to define platform specific capabilities. Each concrete OVAL State is comprised of a set of entities that describe a specific system state.

Property	Type	Count	Description
id	oval:StateIDPattern	1	The globally unique identifier of an OVAL State contained in the OVAL Definitions.
version	unsigned int	1	The version of the globally unique OVAL State referenced by the id property.
operator	oval:OperatorEnumeration	0..1	The value to be used as the operator for the OVAL State, in order to know how to combine the set of entities defined within the concrete OVAL State. Default Value: 'AND'.
comment	oval:NonEmptyStringType	1	A short description of the OVAL State.

deprecated	boolean	0..1	Whether or not the OVAL Object has been deprecated. Default Value: 'false'.
notes	NotesType	0..1	A container for individual notes that describe some aspect of the OVAL State.

signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the content.

Table 20: ObjectType Construct

22. VariablesType

The VariablesType provides a container for one or more OVAL Variables.

Property	Type	Count	Description
variable	VariableType	1..*	A collection of OVAL States.

Table 21: VariablesType Construct

23. VariableType

The VariableType is an abstract OVAL Variable that defines the common properties associated with all OVAL Variables defined in the OVAL Definition Model. The VariableType provides an extension point for concrete OVAL Variables. Concrete OVAL Variables extend this type to provide specific details.

Each concrete OVAL Variable has a collection of values. This collection of values may be the empty set. The proper handling of an empty collection of values for a given variable is left to the context in which the OVAL Variable is used. In some contexts an empty collection of values will be an error, and in other contexts an empty collection of values will be needed for proper evaluation. This context sensitive behavior is defined in [I-D.draft-haynes-sacm-oval-processing-model]. All OVAL Variable values MUST conform to the datatype specified by the datatype property.

Property	Type	Count	Description
id	oval:VariableIDPattern	1	The globally unique identifier of an OVAL Variable contained in the OVAL Definitions.

version	unsigned int	1	The version of the globally unique OVAL Variable referenced by the id property.
datatype	oval:SimpleDatatypeEnumeration	1	The datatype of the value(s) in the OVAL Variable. The 'record' datatype is not supported in OVAL Variables.
comment	oval:NonEmptyStringType	1	The documentation associated with the OVAL Variable instance.
deprecated	boolean	0..1	Whether or not the OVAL Variable has been deprecated. Default Value: 'false'.
signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the content.

Table 22: VariableType Construct

24. external_variable

The external_variable is an extension of the VariableType and provides a way of defining variables whose values come from a source outside of the OVAL Definition.

An external_variable can have any number of possible_value and/or possible_restriction elements in any order.

Property	Type	Count	Description
possible_value	PossibleValueType	0..*	Defines one acceptable value for an external variable.
possible_restriction	PossibleRestrictionType	0..*	Defines a range of acceptable values for an external variable.

Table 23: external_variable Construct

25. PossibleValueType

The PossibleValueType provides a way to explicitly state an acceptable value for an external variable.

Property	Type	Count	Description
hint	string	1	A short description that describes the allowed value.
value	string	1	An acceptable value for the external variable.

Table 24: PossibleValueType Construct

26. PossibleRestrictionType

The PossibleRestrictionType provides a way to explicitly list a range of acceptable values for an external variable. The operation attribute may be used to combine multiple restriction elements using a specified operation. See the Operator Enumeration Evaluation section in [I-D.draft-haynes-sacm-oval-processing-model] for more information on how to combine the individual results.

Property	Type	Count	Description
restriction	RestrictionType	1..*	The restriction that is being applied.
operation	OperationEnumeration	1	The operation to be applied to the restriction. Default Value: 'AND'.
hint	string	1	A short description that describes the allowed value.

Table 25: PossibleRestrictionType Construct

27. RestrictionType

The RestrictionType defines how to describe a restriction for an external variable.

Property	Type	Count	Description
operation	OperationEnumeration	1	The operation to be applied to the restriction. Default Value: 'AND'.
value	string	1	An acceptable value for the external variable.

Table 26: RestrictionType Construct

28. constant_variable

The constant_variable extends the VariableType and provides a way of defining variables whose value is immutable.

Property	Type	Count	Description
value	ValueType	1..*	Defines a value represented by the OVAL Variable.

Table 27: constant_variable Construct

29. ValueType

The ValueType element defines a variable value.

Property	Type	Count	Description
value	string	0..*	Allows any simple type to be used as a value. If no value is specified the value is considered to be the empty string.

Table 28: ValueType Construct

30. local_variable

The local_variable is an extension of the VariableType and provides a way of defining variables whose value is determined by another local OVAL Construct. The value of this variable is determined at evaluation time.

A local_variable can be constructed from a single component or via complex functions to manipulate the referenced components.

Property	Type	Count	Description
components	ComponentsGroup	1..*	The collection of ComponentGroup constructs to be evaluated in the local_variable.

Table 29: local_variable Construct

31. ComponentGroup

The ComponentGroup defines a set of constructs that can be used within a local_variable or OVAL Function. When defining a local_variable or OVAL Function, one or more of these constructs maybe used to specify the desired collection of values for the OVAL Variable.

Property	Type	Count	Description
object_component	ObjectComponentType	0..*	A component of an OVAL Variable whose value comes from an OVAL Object.
variable_component	VariableComponentType	0..*	A component of an OVAL Variable whose value comes from another OVAL Variable.
literal_component	LiteralComponentType	0..*	A component of an OVAL Variable whose value is a literal value.
functions	FunctionGroup	0..*	One or more of a set of functions that act upon one or more components of an OVAL Variable.

Table 30: ComponentGroup Construct

32. LiteralComponentType

The LiteralComponentType defines the way to provide an immutable value to a local_variable.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	0..1	Defines the datatype. Default Value: 'string'.
value	string	0..1	The value of the literal component. If no value is specified the value is considered to be the empty string.

Table 31: LiteralComponentType Construct

33. ObjectComponentType

The ObjectComponentType defines the mechanism for retrieving OVAL Item Entity values, specified by an OVAL Object, to provide one or more values to a component of a local_variable or OVAL Function.

Property	Type	Count	Description
object_ref	oval:ObjectIDPattern	1	Specifies the identifier for the OVAL Object which the component refers.
item_field	oval:NonEmptyStringType	1	The name of the OVAL Item Entity to use for the value(s) of the OVAL Variable.
record_field	oval:NonEmptyStringType	0..1	Allows the

				retrieval of a specified OVAL field to be retrieved from an OVAL Item Entity that has a datatype of 'record'.

Table 32: ObjectComponentType Construct

34. VariableComponentType

The VariableComponentType defines the way to specify that the value(s) of another OVAL Variable should be used as the value(s) for a component of a local_variable or OVAL Function.

A variable component is a component that resolves to the value(s) associated with the referenced OVAL Variable.

Property	Type	Count	Description
var_ref	oval:VariableIDPattern	1	Specifies the Identifier for the OVAL Variable to which the component refers. The var_ref property MUST refer to an existing OVAL Variable. Care must be taken to ensure that the referenced OVAL Variable does not result in a circular reference as it could result in an infinite loop when evaluated.

Table 33: VariableComponentType Construct

35. FunctionGroup

The FunctionGroup defines the possible OVAL Functions for use in OVAL Content to manipulate collected data. OVAL Functions can be nested within one another to achieve the case where one needs to perform multiple functions on a collection of values.

Property	Type	Count	Description
arithmetic	ArithmeticFunctionType	0..1	A function for performing basic math on numbers.
begin	BeginFunctionType	0..1	A function that ensures

				that a collected string starts with a specified string.

concat	ConcatFunctionType	0..1	A function that combines multiple strings.
end	EndFunctionType	0..1	A function that determines whether a collected string ends with a specified string or not.
escape_regex	EscapeRegexFunctionType	0..1	A function that escapes all of the reserved regular expression characters in a string.
split	SplitFunctionType	0..1	A function that splits a string into parts, using a delimeter.
substring	SubstringFunctionType	0..1	A function that creates a substring from a value.
time_difference	TimeDifferenceFunctionType	0..1	A function that calculates the difference

				between two times.
unique	UniqueFunctionType	0..1	A function that takes one or more components and removes any duplicate value from

				the set of components.
regex_capture	RegexCaptureFunctionType	0..1	A function that uses a regular expression to capture a substring of a collected string value.

Table 34: FunctionGroup Construct

36. ArithmeticFunctionType

The ArithmeticFunctionType defines a function that calculates a given, simple mathematic operation between two or more values. This function applies the specified mathematical operation on two or more integer or float values. The result of this operation is a single integer or float value, unless any of the sub-components resolve to multiple values, in which case the result will be an array of values, corresponding to the arithmetic operation applied to the Cartesian product of the values.

In the case of mixed integers and floats, the result will be a float value.

Property	Type	Count	Description
arithmetic_operation	ArithmeticEnumeration	1	The operation to perform.
values	ComponentGroup	2..*	Any type from the ComponentGroup.

Table 35: ArithmeticFunctionType Construct

37. BeginFunctionType

The BeginFunctionType defines a function that ensures that the specified values start with a specified character or string. This function operates on a single sub-component of datatype string and ensures that the specified value(s) start with the characters specified in the character property. When a value does not start with the specified characters, the function will prepend add the complete set of characters from the character property to the string. Otherwise, the string value will remain unchanged.

Property	Type	Count	Description
character	string	1	The character or string to use for the function.
value	ComponentGroup	1	Any type from the ComponentGroup.

Table 36: BeginFunctionType Construct

38. ConcatFunctionType

The ConcatFunctionType defines a function that concatenates the values specified together into a single string value. This function combines the values of two or more sub-components into a single string value. The function combines the sub-component values in the order that they are specified. That is, the first sub-component specified will always be at the beginning of the newly created string value and the last sub-component will always be at the end of the newly created string value.

Property	Type	Count	Description
values	ComponentGroup	2..*	Any type from the ComponentGroup.

Table 37: ConcatFunctionType Construct

39. CountFunctionType

The CountFunctionType defines a function that counts the values represented by one or more components as an integer. This function determines the total number of values referenced by all of the specified sub-components.

Property	Type	Count	Description
values	ComponentGroup	1..*	Any type from the ComponentGroup.

Table 38: CountFunctionType Construct

40. EndFunctionType

The EndFunctionType defines a function that ensures that the specified values end with a specified character or string. This function operates on a single sub-component of datatype string and ensures that the specified value(s) end with the characters specified in the character property. When a value does not end with the specified characters, the function will add the complete set of characters from the character property to the end of the string. Otherwise, the string value will remain unchanged.

Property	Type	Count	Description
character	string	1	The character or string to use for the function.
value	ComponentGroup	1	Any type from the ComponentGroup.

Table 39: EndFunctionType Construct

41. EscapeRegexFunctionType

The EscapeRegexFunctionType defines a function that escapes all of the regular expression reserved characters in a given string. This function operates on a single sub-component, escaping reserved

regular expression characters for each sub-component value. The set of metacharacters, in the Perl 5 regular expression syntax, which must be escaped for this purpose is as follows, enclosed by single quotes: '^$\.[](){}*+?|'. Please see the Regular Expression Support section in [I-D.draft-cokus-sacm-oval-common-model] for more information on the Perl 5 regular expression syntax that is supported in the OVAL Language.

Property	Type	Count	Description
value	ComponentGroup	1	Any type from the ComponentGroup.

Table 40: EscapeRegexFunctionType Construct

42. SplitFunctionType

The SplitFunctionType defines a function that splits a string value into multiple values, based on a specified delimiter. This function operates on a single sub-component and results in an array of values, where each values is the splitting the subject string using the specified delimiter.

If the sub-component being split includes a string that either begins with or ends with the delimiter, there will be an empty string value included either at the beginning or end, respectively.

If multiple instances of the delimiter appear consecutively, each instance will result in an additional empty string value.

If the delimiter is not found in the subject string, the entire subject string will be included in the result.

Property	Type	Count	Description
delimiter	string	1	The string to use as a delimiter.
value	ComponentGroup	1	Any type from the ComponentGroup.

Table 41: SplitFunctionType Construct

43. SubstringFunctionType

The SubstringFunctionType defines a function that takes a string value and produces a value that contains a portion of the original string.

Property	Type	Count	Description
substring_start	int	1	The starting index to use for the substring. This property is 1-based, meaning that a value of 1 represents the first character of the subject string. A value less than 1 is also interpreted as the first character in the subject string. If the substring_start property exceeds the length of the subject string an error MUST be reported.
substring_length	int	1	Represents the length of the substring to be taken from the source string, including the starting character. Any substring_length that exceeds the length of the string or is negative indicates to include all characters from the starting character until the end of the source string.
value	ComponentGroup	1	Any type from the ComponentGroup.

Table 42: SubstringFunctionType Construct

44. TimeDifferenceFunctionType

The TimeDifferenceFunctionType defines a function that produces a value containing the difference in seconds between two date-time values. If a single sub-component is specified, then the time difference is between the specified date-time and the current date-time. The current time is the time at which the function is evaluated. If two sub-components are specified, then the difference is that between the two specified date-times.

Property	Type	Count	Description
format_1	DateTimeFormatEnumeration	0..1	The format for the first date-time value specified. Note: If specifying a single value, use format_1 to specify the implied current date-time. Default Value: 'year_month_day'.
format_2	DateTimeFormatEnumeration	0..1	The format for the second date-time value specified. Note: If specifying a single value, use format_2 to specify the value's format, as format_1 is used for the implied current date-time. Default Value: 'year_month_day'.
value	ComponentGroup	1..2	Any type from the ComponentGroup.

Table 43: TimeDifferenceFunctionType Construct

If a sub-component value does not conform to the format specified in the DateTimeFormatEnumeration an error MUST be reported.

The datatype associated with the sub-components MUST be 'string' or 'int' depending on which date time format is specified. The result of this function is always an integer. The following table states which datatype MUST be used with which format from the DateTimeFormatEnumeration.

Value	Description
year_month_day	string
month_day_year	string
day_month_year	string
win_filetime	int
seconds_since_epoch	int

Table 44: DateTimeFormat Datatype Enumeration Table

45. UniqueFunctionType

The UniqueFunctionType defines a function that removes any duplicate value from the set of values represented by one or more components. This function takes one or more sub-components and removes any duplicate values across the sub-components. A duplicate value is defined as any value that is equal to another value when compared as a string value.

Property	Type	Count	Description
values	ComponentGroup	1..*	Any type from the ComponentGroup

Table 45: UniqueFunctionType Construct

46. RegexCaptureFunctionType

The RegexCaptureFunctionType defines a function operating on a single component, which extracts a substring from each of its values.

The pattern property specifies a regular expression, which SHOULD contain a single capturing sub-pattern (using parentheses). If the regular expression contains multiple capturing sub-patterns, only the first capture is used. If there are no capturing sub-patterns, the result for each target string MUST be the empty string. Otherwise, if the regular expression could match the target string in more than one place, only the first match (and its first capture) is used. If no matches are found in a target string, the result for that target MUST be the empty string.

Note that a quantified capturing sub-pattern does not produce multiple substrings. Standard regular expression semantics are such that if a capturing sub-pattern is required to match multiple times in order for the overall regular expression to match, the capture produced is the last substring to have matched the sub-pattern.

If any of the Perl 5 regular expression syntax metacharacters are to be used literally, then they must be escaped. The set of metacharacters which must be escaped for this purpose is as follows, enclosed by single quotes: '^$\.[]{}*+?|'. Please see the Regular Expression Support section in [I-D.draft-cokus-sacm-oval-common-model] for more information on the Perl 5 regular expression syntax that is supported in the OVAL Language.

Property	Type	Count	Description
pattern	string	1	The string to use as a regular expression pattern.
value	ComponentGroup	1	Any type from the ComponentGroup.

Table 46: RegexCaptureFunctionType Construct

47. ArithmeticEnumeration

The ArithmeticEnumeration defines an enumeration for the possible values for the arithmetic function.

Value	Description
add	Indicates addition.
multiply	Indicates multiplication.

Table 47: Arithmetic Enumeration

48. DateTimeFormatEnumeration

The DateTimeFormatEnumeration defines an enumeration for the possible values for the date-time values.

Value	Description
year_month_day	This value indicates a format that follows the following patterns: o yyyymmdd o yyyymmddThhmmss o yyyy/mm/dd hh:mm:ss o yyyy/mm/dd o yyyy-mm-dd hh:mm:ss o yyyy-mm-dd
month_day_year	This value indicates a format that follows the following patterns: o mm/dd/yyyy hh:mm:ss o mm/dd/yyyy

		o mm-dd-yyyy hh:mm:ss o mm-dd-yyyy o NameOfMonth, dd yyyy hh:mm:ss o NameOfMonth, dd yyyy o AbreviatedNameOfMonth, dd yyyy hh:mm:ss o AbreviatedNameOfMonth, dd yyyy
day_month_year	This value indicates a format that follows the following patterns: o dd/mm/yyyy hh:mm:ss o dd/mm/yyyy o dd-mm-yyyy hh:mm:ss o dd-mm-yyyy	
win_filetime	This value indicates a date-time that follows the windows file time format [WIN-FILETIME].	
seconds_since_epoch	This value indicates a date-time that represents the time in seconds since the UNIX Epoch. The UNIX epoch is the time 00:00:00 UTC on January 1, 1970.	

Figure 1: DateTimeFormat Enumeration

49. FilterActionEnumeration

The FilterActionEnumeration defines an enumeration for the possible values for filtering a set of items.

Value	Description
include	A value that indicates to include matching items from the set.
exclude	A value that indicates to exclude matching items from the set.

Table 48: FilterAction Enumeration

50. SetOperatorEnumeration

The SetOperatorEnumeration defines an enumeration for the possible values defining a set.

Value	Description
COMPLEMENT	A value that indicates to include only the elements from the first set that are not found in the second.
INTERSECTION	A value that indicates to include all of the values common to both sets.
UNION	A value that indicates to include all values found in either of the sets.

Table 49: SetOperator Enumeration

51. EntityAttributeGroup

The EntityAttributeGroup defines a set of attributes that are common to all OVAL Object and OVAL State entities.

Some OVAL Entities provide additional restrictions on these attributes and their allowed values.

Property	Type	Count	Description
datatype	oval:DatatypeEnumeration	0..1	The datatype for the entity. Default Value: 'string'.
operation	oval:OperationEnumeration	0..1	The operation that is to be performed on the entity. Default Value: 'equals'.
mask	boolean	0..1	Tells the data collection that this entity contains sensitive data. Data marked with mask='true' should be used only in the evaluation, and not be included in the results. Default Value: 'false'.
var_ref	oval:VariableIDPattern	0..1	Points to a variable Identifier within the OVAL document which should be used to calculate the entity's value.
var_check	oval:CheckEnumeration	0..1	Directs how to either collect data or evaluate state for the entity.

Table 50: EntityAttributeGroup

52. EntitySimpleBaseType

The EntitySimpleBaseType is an abstract type that defines a base type for all simple entities. Entities represent the individual properties for OVAL Objects and OVAL States.

Property	Type	Count	Description
attributes	EntityAttributeGroup	1	The standard attributes available to all entities.

value	String	0..1	The value of the entity. An empty string value MUST be used when referencing an OVAL Variable.

Table 51: EntitySimpleBaseType Construct

53. EntityComplexBaseType

The EntityComplexBaseType is an abstract type that defines a base type for all complex entities. Entities represent the individual properties for OVAL Objects and OVAL States.

Property	Type	Count	Description
attributes	EntityAttributeGroup	1	The standard attributes available to all entities.

Table 52: EntityComplexBaseType Construct

54. EntityObjectIPAddressType

The EntityObjectIPAddressType extends the EntitySimpleBaseType and describes an IPv4 or IPv6 IP address.

Property	Type	Count	Description
datatype	oval: SimpleDatatype Enumeration	1	Possible values: o 'ipv4_address' o 'ipv6_address' Also allows an empty string value.

Figure 2: EntityObjectIPAddressType Construct

55. EntityObjectIPAddressStringType

The EntityObjectIPAddressStringType extends the EntitySimpleBaseType and describes an IPv4 or IPv6 IP address or a string representation of the address.

Property	Type	Count	Description
datatype	oval: SimpleDatatype Enumeration	1	Possible values: o 'ipv4_address' o 'ipv6_address' o 'string' Also allows an empty string

			value.

Figure 3: EntityObjectIPAddressStringType Construct

56. EntityObjectAnySimpleType

The EntityObjectAnySimpleType extends the EntitySimpleBaseType and describes any simple data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	Any simple datatype. Also allows an empty string value.

Table 53: EntityObjectAnySimpleType Construct

57. EntityObjectBinaryType

The EntityObjectBinaryType extends the EntitySimpleBaseType and describes any simple binary data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'binary'. Also allows an empty string value.

Table 54: EntityObjectBinaryType Construct

58. EntityObjectBoolType

The EntityObjectBoolType extends the EntitySimpleBaseType and describes any simple boolean data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'boolean'. Also allows an empty string value.

Table 55: EntityObjectBoolType Construct

59. EntityObjectFloatType

The EntityObjectFloatType extends the EntitySimpleBaseType and describes any simple float data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'float'. Also allows an empty string value.

Table 56: EntityObjectFloatType Construct

60. EntityObjectIntType

The EntityObjectIntType extends the EntitySimpleBaseType and describes any simple integer data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'int'. Also allows an empty string value.

Table 57: EntityObjectIntType Construct

61. EntityObjectStringType

The EntityObjectStringType extends the EntitySimpleBaseType and describes any simple string data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'string'. Also allows an empty string value.

Table 58: EntityObjectStringType Construct

62. EntityObjectVersionType

The EntityObjectVersionType extends the EntitySimpleBaseType and describes any simple version data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'version'. Also allows an empty string value.

Table 59: EntityObjectVersionType Construct

63. EntityObjectRecordType

The EntityObjectRecordType extends the EntityComplexBaseType and allows assertions to be made on entities with uniquely named fields. It is intended to be used to assess the results of things such as SQL statements and similar data.

Property	Type	Count	Description
datatype	oval:ComplexDatatypeEnumeration	1	This value is fixed as 'record'.
operation	oval:OperationEnumeration	0..1	This value is fixed as 'equals'.
mask	boolean	0..1	Tells the data collection that this entity contains sensitive data. Data marked with mask='true' should be used only in the evaluation, and not be included in the results. Note that when the mask property is set to 'true', all child field elements must be masked regardless of the child field's mask attribute value. Default Value: 'false'.
var_ref	oval:VariableIDPattern	0..1	Use of this property is prohibited.
var_check	oval:CheckEnumeration	0..1	Use of this property is prohibited.

Table 60: EntityObjectRecordType Construct

64. EntityObjectFieldType

The EntityObjectFieldType defines an entity type that captures the details of a single field for a record.

Property	Type	Count	Description
attributes	EntityAttributeGroup	1	The standard attributes available to all entities.
name	string	1	The name of the field. Names MUST be all lower case characters in the range of a-z. Names MUST be unique within a record.
value	string	0..1	The value of the field. An empty string value MUST be used when referencing an OVAL Variable.

Table 61: EntityObjectFieldType Construct

65. EntityStateSimpleBaseType

The EntityStateSimpleBaseType extends the EntitySimpleBaseType and defines a simple base type for OVAL States.

Property	Type	Count	Description
entity_check	oval:CheckEnumeration	0..1	Defines how to handle multiple item entities with the same name. Default Value: 'all'.
value	string	0..1	The value of the entity. An empty string value MUST be used when referencing an OVAL Variable.

Table 62: EntityStateSimpleBaseType Construct

66. EntityStateComplexBaseType

The EntityStateComplexBaseType extends the EntityComplexBaseType defines a complex base type for OVAL States.

Property	Type	Count	Description
entity_check	oval:CheckEnumeration	0..1	Defines how to handle multiple item entities with the same name. Default Value: 'all'.

Table 63: EntityStateComplexBaseType Construct

67. EntityStateIPAddressType

The EntityStateIPAddressType extends the EntityStateSimpleBaseType and describes an IPv4 or IPv6 IP address.

Property	Type	Count	Description
datatype	oval: SimpleDatatype Enumeration	1	Possible values: o 'ipv4_address' o 'ipv6_address' Also allows an empty string value.

Figure 4: EntityStateIPAddressType Construct

68. EntityStateIPAddressStringType

The EntityStateIPAddressStringType extends the EntityStateSimpleBaseType and describes an IPv4 or IPv6 IP address or a string representation of the address.

Property	Type	Count	Description
datatype	oval: SimpleDatatype Enumeration	1	Possible values: o 'ipv4_address' o 'ipv6_address' o 'string' Also allows an empty string value.

Figure 5: EntityStateIPAddressStringType Construct

69. EntityStateAnySimpleType

The EntityStateAnySimpleType extends the EntityStateSimpleBaseType and describes any simple data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	Any simple datatype. Also allows an empty string value.

Table 64: EntityStateAnySimpleType Construct

70. EntityStateBinaryType

The EntityStateBinaryType extends the EntityStateSimpleBaseType and describes any simple binary data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'binary'. Also allows an empty string value.

Table 65: EntityStateBinaryType Construct

71. EntityStateBoolType

The EntityStateBoolType extends the EntityStateSimpleBaseType and describes any simple boolean data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'boolean'. Also allows an empty string value.

Table 66: EntityStateBoolType Construct

72. EntityStateFloatType

The EntityStateFloatType extends the EntityStateSimpleBaseType and describes any simple float data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'float'. Also allows an empty string value.

Table 67: EntityStateFloatType Construct

73. EntityStateIntType

The EntityStateIntType extends the EntityStateSimpleBaseType and describes any simple integer data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'int'. Also allows an empty string value.

Table 68: EntityStateIntType Construct

74. EntityStateEVRStringType

The EntityStateEVRStringType extends the EntityStateSimpleBaseType and describes an EPOCH:VERSION-RELEASE string data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'evr_string'. Also allows an empty string value.

Table 69: EntityStateEVRStringType Construct

75. EntityStateDebianEVRStringType

The EntityStateDebianEVRStringType extends the EntityStateSimpleBaseType and describes an EPOCH:UPSTREAM_VERSION-DEBIAN_REVISION string data for a Debian package.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'debian_evr_string'. Also allows an empty string value.

Table 70: EntityStateDebianEVRStringType Construct

76. EntityStateVersionType

The EntityStateVersionType extends the EntityStateSimpleBaseType and describes a version string data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'version'. Also allows an empty string value.

Table 71: EntityStateVersionType Construct

77. EntityStateFileSetRevisionType

The EntityStateFileSetRevisionType extends the EntityStateSimpleBaseType and describes a file set revision string data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'fileset_revision'. Also allows an empty string value.

Table 72: EntityStateFileSetRevisionType Construct

78. EntityStateIOSVersionType

The EntityStateIOSVersionType extends the EntityStateSimpleBaseType and describes a Cisco IOS version string data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	Possible values: o 'ios_version' o 'string' The string type is an option in

				order to allow use of regular expressions.

Figure 6: EntityStateIOSVersionType Construct

79. EntityStateStringType

The EntityStateStringType extends the EntitySimpleBaseType and describes any simple string data.

Property	Type	Count	Description
datatype	oval:SimpleDatatypeEnumeration	1	This value is fixed as 'string'. Also allows an empty string value.

Table 73: EntityStateStringType Construct

80. EntityStateRecordType

The EntityStateRecordType extends the EntityStateComplexBaseType and allows assertions to be made on entities with uniquely named fields. It is intended to be used to assess the results of things such as SQL statements and similar data.

Property	Type	Count	Description

datatype	oval:ComplexDatatypeEnumeration	1	This value is fixed as 'record'.
operation	oval:OperationEnumeration	0..1	This value is fixed as 'equals'.
mask	boolean	0..1	Tells the data collection that this entity contains sensitive data. Data marked with mask='true' should be used only in the evaluation, and not be included in the results. Note that when the mask property is set to 'true', all child field elements must be masked regardless of the

				child field's mask attribute value. Default Value: 'false'.
var_ref	oval:VariableIDPattern	0..1	Use of this	

				property is prohibited.
var_check	oval:CheckEnumeration	0..1	Use of this property is prohibited.	

Table 74: EntityStateRecordType Construct

81. EntityStateFieldType

The EntityStateFieldType defines an entity type that captures the details of a single field for a record.

Property	Type	Count	Description
attributes	EntityAttributeGroup	1	The standard attributes available to all entities.
name	string	1	The name of the field. Names MUST be all lower case characters in the range of a-z. Names MUST be unique within a record.
value	string	0..1	The value of the field. An empty string value MUST be used when referencing an OVAL Variable.

Table 75: EntityStateFieldType Construct

82. OVAL Definitions Model Schema

The following XML Schema implements the OVAL Definitions Model.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:oval="http://oval.mitre.org/XMLSchema/oval-common-5"
```



```
  xmlns:oval-def="http://oval.mitre.org/XMLSchema/
  oval-definitions-5"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron"
  targetNamespace="http://oval.mitre.org/XMLSchema/
  oval-definitions-5"
  elementFormDefault="qualified" version="5.11">
```

```
    <xsd:import
      namespace="http://oval.mitre.org/XMLSchema/oval-common-5"
      schemaLocation="oval-common-schema.xsd"/>
    <xsd:import
      namespace="http://www.w3.org/2000/09/xmldsig#"
      schemaLocation="xmldsig-core-schema.xsd"/>
    <xsd:annotation>
      <xsd:documentation>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.</xsd:documentation>
      <xsd:appinfo>
        <schema>Core Definition</schema>
        <version>5.11.1</version>
        <date>4/22/2015 09:00:00 AM</date>
        <terms_of_use>Copyright (C) 2010 United States Government.
          All Rights Reserved.</terms_of_use>
        <sch:ns prefix="oval-def"
          uri="http://oval.mitre.org/XMLSchema/
          oval-definitions-5"/>
        <sch:ns prefix="xsi"
          uri="http://www.w3.org/2001/XMLSchema-instance"
        />
      </xsd:appinfo>
    </xsd:annotation>
  <!-- ======================================================== -->
  <!-- ======================================================== -->
  <!-- ======================================================== -->
```



```
    <xsd:element name="oval_definitions">
      <xsd:annotation>
        <xsd:documentation>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.</xsd:documentation>
        <xsd:appinfo>
          <sch:pattern id="oval-def_empty_def_doc">
            <sch:rule
              context="oval-def:oval_definitions">
              <sch:assert
                test="oval-def:definitions or oval-def:tests or
                oval-def:objects or oval-def:states or
                oval-def:variables"
                >A valid OVAL Definition document
                must contain at least one
                definitions, tests, objects, states,
                or variables element. The optional
                definitions, tests, objects, states,
                and variables sections define the
                specific characteristics that should
                be evaluated on a system to
                determine the truth values of the
                OVAL Definition Document. To be
                valid though, at least one
                definitions, tests, objects, states,
                or variables element must be
                present.</sch:assert>
```

```
            </sch:rule>
          </sch:pattern>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="generator"
            type="oval:GeneratorType">
            <xsd:annotation>
              <xsd:documentation>The required
                generator section provides
                information about when the
                definition file was compiled and
                under what
                version.</xsd:documentation>
            </xsd:annotation>
```



```
          </xsd:element>
          <xsd:element name="definitions"
            type="oval-def:DefinitionsType"
            minOccurs="0" maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>The optional
                definitions section contains 1 or
                more
                definitions.</xsd:documentation>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="tests"
            type="oval-def:TestsType" minOccurs="0"
            maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>The optional tests
                section contains 1 or more
                tests.</xsd:documentation>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="objects"
            type="oval-def:ObjectsType"
            minOccurs="0" maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>The optional
                objects section contains 1 or more
                objects.</xsd:documentation>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="states"
            type="oval-def:StatesType" minOccurs="0"
            maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>The optional states
                section contains 1 or more
                states.</xsd:documentation>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="variables"
            type="oval-def:VariablesType"
            minOccurs="0" maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>The optional
                variables section contains 1 or more
                variables.</xsd:documentation>
            </xsd:annotation>
          </xsd:element>
          <xsd:element ref="ds:Signature"
```



```
            minOccurs="0" maxOccurs="1">
```

```
            <xsd:annotation>
              <xsd:documentation>The optional
                Signature element allows an XML
                Signature as defined by the W3C to
                be attached to the document. This
                allows authentication and data
                integrity to be provided to the
                user. Enveloped signatures are
                supported. More information about
                the official W3C Recommendation
                regarding XML digital signatures can
                be found at
                http://www.w3.org/TR/xmldsig-core/.
              </xsd:documentation>
            </xsd:annotation>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:key name="definitionKey">
        <xsd:annotation>
          <xsd:documentation>Enforce uniqueness
            amongst the ids differentiating the
            individual definition
            elements.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector
          xpath="oval-def:definitions/oval-def:definition"/>
        <xsd:field xpath="@id"/>
      </xsd:key>
      <xsd:key name="testKey">
        <xsd:annotation>
          <xsd:documentation>Enforce uniqueness
            amongst the ids differentiating the
            individual test
            elements.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector xpath="oval-def:tests/*"/>
        <xsd:field xpath="@id"/>
      </xsd:key>
      <xsd:key name="objectKey">
        <xsd:annotation>
          <xsd:documentation>Enforce uniqueness
            amongst the ids differentiating the
            individual object
            elements.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector xpath="oval-def:objects/*"/>
```



```
        <xsd:field xpath="@id"/>
      </xsd:key>
      <xsd:key name="stateKey">
        <xsd:annotation>
          <xsd:documentation>Enforce uniqueness
            amongst the ids differentiating the
            individual state
            elements.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector xpath="oval-def:states/*"/>
        <xsd:field xpath="@id"/>
      </xsd:key>
      <xsd:key name="variableKey">
        <xsd:annotation>
          <xsd:documentation>Enforce uniqueness
            amongst the ids differentiating the
            individual variable
            elements.</xsd:documentation>
        </xsd:annotation>
        <xsd:selector xpath="oval-def:variables/*"/>
        <xsd:field xpath="@id"/>
      </xsd:key>
      <xsd:keyref name="extendKeyRef"
        refer="oval-def:definitionKey">
        <xsd:annotation>
          <xsd:documentation>Requires each
```

```
            definition reference to refer to a valid
            definition id.</xsd:documentation>
          </xsd:annotation>
          <xsd:selector xpath=".//*"/>
          <xsd:field xpath="@definition_ref"/>
        </xsd:keyref>
        <xsd:keyref name="testKeyRef"
          refer="oval-def:testKey">
          <xsd:annotation>
            <xsd:documentation>Requires each test
              reference to refer to a valid test
              id.</xsd:documentation>
          </xsd:annotation>
          <xsd:selector xpath=".//*"/>
          <xsd:field xpath="@test_ref"/>
        </xsd:keyref>
        <xsd:keyref name="objectKeyRef"
          refer="oval-def:objectKey">
          <xsd:annotation>
            <xsd:documentation>Requires each object
              reference to refer to a valid object
              id.</xsd:documentation>
```



```
          </xsd:annotation>
          <xsd:selector xpath=".//*"/>
          <xsd:field xpath="@object_ref"/>
        </xsd:keyref>
        <xsd:keyref name="stateKeyRef"
          refer="oval-def:stateKey">
          <xsd:annotation>
            <xsd:documentation>Requires each state
              reference to refer to a valid state
              id.</xsd:documentation>
          </xsd:annotation>
          <xsd:selector xpath=".//*"/>
          <xsd:field xpath="@state_ref"/>
        </xsd:keyref>
        <xsd:keyref name="variableKeyRef"
          refer="oval-def:variableKey">
          <xsd:annotation>
            <xsd:documentation>Requires each variable
              reference to refer to a valid variable
              id.</xsd:documentation>
          </xsd:annotation>
          <xsd:selector xpath=".//*"/>
          <xsd:field xpath="@var_ref"/>
        </xsd:keyref>
        <xsd:keyref name="object_referenceKeyRef"
          refer="oval-def:objectKey">
          <xsd:annotation>
            <xsd:documentation>Require each object
              reference in a set element to refer to a
              valid object id.</xsd:documentation>
          </xsd:annotation>
          <xsd:selector
            xpath=".//oval-def:object_reference"/>
          <xsd:field xpath="."/>
        </xsd:keyref>
        <xsd:keyref name="filterKeyRef"
          refer="oval-def:stateKey">
          <xsd:annotation>
            <xsd:documentation>Require each filter in
              a set element to refer to a valid state
              id.</xsd:documentation>
          </xsd:annotation>
          <xsd:selector xpath=".//oval-def:filter"/>
          <xsd:field xpath="."/>
        </xsd:keyref>
      </xsd:element>
      <xsd:element name="notes"
        substitutionGroup="oval:notes">
```

```
      <xsd:annotation>
        <xsd:documentation>The notes element is a
          container for one or more note child
          elements. It exists for
          backwards-compatibility purposes, for the
          pre-5.11.0 oval-def:NotesType, which has
          been replaced by the oval:notes element in
          5.11.1.</xsd:documentation>
        <xsd:appinfo>
          <oval:deprecated_info>
            <oval:version>5.11.1</oval:version>
            <oval:reason>Replaced by the oval:notes
              element.</oval:reason>
            <oval:comment>This object has been
              deprecated and may be removed in a
              future version of the
              language.</oval:comment>
          </oval:deprecated_info>
          <sch:pattern id="oval_def_notes_dep">
            <sch:rule context="oval-def:notes">
              <sch:report test="true()">DEPRECATED
                ELEMENT: <sch:value-of
                  select="name()"/> parent ID:
                  <sch:value-of select="../@id"
                /></sch:report>
            </sch:rule>
          </sch:pattern>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension base="oval:NotesType">
            <xsd:sequence>
              <xsd:element name="note"
                type="xsd:string" minOccurs="0"
                maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
  <!-- ======================================================== -->
  <!-- ==================  GENERATOR  ======================== -->
  <!-- ======================================================== -->
    <!--
  The GeneratorType is defined by the oval common schema.
  Please refer to that documentation for a description of the
  complex type.
```



```
           -->
  <!-- ======================================================== -->
  <!-- ==================  DEFINITIONS  ======================= -->
  <!-- ======================================================== -->
    <xsd:complexType name="DefinitionsType">
      <xsd:annotation>
        <xsd:documentation>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.</xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element ref="oval-def:definition"
          minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
```

```
    <xsd:element name="definition"
      type="oval-def:DefinitionType">
      <xsd:annotation>
        <xsd:documentation>The definition element
          represents the globally defined element of
          type DefinitionType. For more information
          please see the documentation on the
          DefinitionType.</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:complexType name="DefinitionType">
      <xsd:annotation>
        <xsd:documentation>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.</xsd:documentation>
        <xsd:documentation>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
          oval: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. The class gives a hint to a user
          so they can know what the definition
          writer is trying to say. See the
          definition of oval-def: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.</xsd:documentation>
        <xsd:documentation>When the deprecated
          attribute is set to true, the definition
          is considered to be deprecated. The
          criteria child element of a deprecated
          definition is optional. If a deprecated
          definition does not contain a criteria
          child element, the definition must
          evaluate to "not evaluated". If a
          deprecated definition contains a criteria
          child element, an interpreter should
          evaluate the definition as if it were not
          deprecated, but an interpreter may
          evaluate the definition to "not
          evaluated".</xsd:documentation>
```

```
          <xsd:appinfo>
            <sch:pattern
              id="oval-def_required_criteria">
```



```
              <sch:rule
                context="oval-def:oval_definitions/
                oval-def:definitions/
                oval-def:definition[(@deprecated='false' or
                @deprecated='0') or not(@deprecated)]">
                <sch:assert test="oval-def:criteria">A
                  valid OVAL Definition must contain a
                  criteria unless the definition is a
                  deprecated definition.</sch:assert>
              </sch:rule>
            </sch:pattern>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element ref="ds:Signature"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element name="metadata"
            type="oval-def:MetadataType">
            <xsd:unique name="UniqueAffectedFamily">
              <xsd:annotation>
                <xsd:documentation>Each affected
                  element must have a unique family
                  attribute value.</xsd:documentation>
              </xsd:annotation>
              <xsd:selector xpath="oval-def:affected"/>
              <xsd:field xpath="@family"/>
            </xsd:unique>
          </xsd:element>
          <xsd:element ref="oval:notes" minOccurs="0"
            maxOccurs="1"/>
          <xsd:element name="criteria"
            type="oval-def:CriteriaType" minOccurs="0"
            maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id"
          type="oval:DefinitionIDPattern"
          use="required"/>
        <xsd:attribute name="version"
          type="xsd:nonNegativeInteger" use="required"/>
        <xsd:attribute name="class"
          type="oval:ClassEnumeration" use="required"/>
        <xsd:attribute name="deprecated"
          type="xsd:boolean" use="optional"
          default="false"/>
      </xsd:complexType>
      <xsd:complexType name="MetadataType">
        <xsd:annotation>
          <xsd:documentation>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.</xsd:documentation>
          <xsd:documentation>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.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element name="title" type="xsd:string"/>
          <xsd:element name="affected"
            type="oval-def:AffectedType" minOccurs="0"
            maxOccurs="unbounded">
            <xsd:unique name="UniqueAffectedPlatform">
              <xsd:annotation>
                <xsd:documentation>Each affected
                  platform element must have a unique
                  value.</xsd:documentation>
              </xsd:annotation>
              <xsd:selector xpath="oval-def:platform"/>
              <xsd:field xpath="."/>
            </xsd:unique>
            <xsd:unique name="UniqueAffectedProduct">
```



```
              <xsd:annotation>
                <xsd:documentation>Each affected
                  product element must have a unique
                  value.</xsd:documentation>
              </xsd:annotation>
              <xsd:selector xpath="oval-def:product"/>
              <xsd:field xpath="."/>
            </xsd:unique>
          </xsd:element>
          <xsd:element name="reference"
            type="oval-def:ReferenceType"
            minOccurs="0" maxOccurs="unbounded"/>
          <xsd:element name="description"
            type="xsd:string"/>
          <xsd:any minOccurs="0" maxOccurs="unbounded"
            processContents="lax"/>
    <!-- For the next major release of OVAL, the xsd:any
         tag above will be modified to only allow elements
         from namespaces other than the default namespace.
         This fixes a bug in the current schema where the
         affected or reference element can appear after the
         description element and still produce a vailid document.
         <xsd:any minOccurs="0" maxOccurs="unbounded"
         namespace="##other" processContents="lax"/>
      -->
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="AffectedType">
        <xsd:annotation>
          <xsd:documentation>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 preselect 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.</xsd:documentation>
          <xsd:documentation>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' might be used to refer to the
            same product. The current convention is to
            fully spell out all terms, and avoid the
            use of abbreviations at all
            costs.</xsd:documentation>
          <xsd:documentation>Please note that the
            AffectedType will change in future
            versions of OVAL in order to support the
            Common Platform Enumeration
            (CPE).</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element name="platform"
            type="xsd:string" minOccurs="0"
            maxOccurs="unbounded"/>
          <xsd:element name="product"
            type="xsd:string" minOccurs="0"
            maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="family"
          type="oval:FamilyEnumeration" use="required"
        />
      </xsd:complexType>
      <xsd:complexType name="ReferenceType">
        <xsd:annotation>
          <xsd:documentation>The ReferenceType complex
            type links the OVAL Definition to a
```



```
            definitive external reference. For
            example, CVE Identifiers are used for
            referencing 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.</xsd:documentation>
```

```
          <xsd:documentation>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.</xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="source" type="xsd:string"
          use="required"/>
        <xsd:attribute name="ref_id" type="xsd:string"
          use="required"/>
        <xsd:attribute name="ref_url"
          type="xsd:anyURI" use="optional"/>
      </xsd:complexType>
      <xsd:complexType name="CriteriaType">
        <xsd:annotation>
          <xsd:documentation>The CriteriaType complex
            type describes a container for a set of
            sub criteria, criteria, criterion, or
            extend_definition elements 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.</xsd:documentation>
          <xsd:documentation>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 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.</xsd:documentation>
          <xsd:documentation>The optional
            applicability_check attribute provides a
            Boolean flag that when true indicates that
            the criteria is being used to determine
            whether the OVAL Definition applies to a
            given system.</xsd:documentation>
        </xsd:annotation>
        <xsd:choice minOccurs="1"
          maxOccurs="unbounded">
          <xsd:element name="criteria"
            type="oval-def:CriteriaType"/>
          <xsd:element name="criterion"
            type="oval-def:CriterionType"/>
          <xsd:element name="extend_definition"
            type="oval-def:ExtendDefinitionType"/>
        </xsd:choice>
        <xsd:attribute name="applicability_check"
          type="xsd:boolean" use="optional"/>
        <xsd:attribute name="operator"
          type="oval:OperatorEnumeration"
          use="optional" default="AND"/>
        <xsd:attribute name="negate"
          type="xsd:boolean" use="optional"
          default="false"/>
        <xsd:attribute name="comment"
          type="oval:NonEmptyStringType"
          use="optional"/>
```

```
    </xsd:complexType>
    <xsd:complexType name="CriterionType">
      <xsd:annotation>
        <xsd:documentation>The CriterionType complex
          type identifies a specific test to be
          included in the definition's
          criteria.</xsd:documentation>
        <xsd:documentation>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.</xsd:documentation>
        <xsd:documentation>The optional
          applicability_check attribute provides a
          Boolean flag that when true indicates that
          the criterion is being used to determine
          whether the OVAL Definition applies to a
          given system.</xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="applicability_check"
        type="xsd:boolean" use="optional"/>
      <xsd:attribute name="test_ref"
        type="oval:TestIDPattern" use="required"/>
      <xsd:attribute name="negate"
        type="xsd:boolean" use="optional"
        default="false"/>
      <xsd:attribute name="comment"
        type="oval:NonEmptyStringType"
        use="optional"/>
    </xsd:complexType>
    <xsd:complexType name="ExtendDefinitionType">
      <xsd:annotation>
        <xsd:documentation>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.</xsd:documentation>
        <xsd:documentation>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 certainsoftware 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.</xsd:documentation>
```



```
        <xsd:documentation>The optional
          applicability_check attribute provides a
```

```
            Boolean flag that when true indicates that
            the extend_definition is being used to
            determine whether the OVAL Definition
            applies to a given
            system.</xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="applicability_check"
          type="xsd:boolean" use="optional"/>
        <xsd:attribute name="definition_ref"
          type="oval:DefinitionIDPattern"
          use="required"/>
        <xsd:attribute name="negate"
          type="xsd:boolean" use="optional"
          default="false"/>
        <xsd:attribute name="comment"
          type="oval:NonEmptyStringType"
          use="optional"/>
      </xsd:complexType>
    <!-- ================================================== -->
    <!-- ====================  TESTS  ===================== -->
    <!-- ================================================== -->
      <xsd:complexType name="TestsType">
        <xsd:annotation>
          <xsd:documentation>The TestsType 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.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element ref="oval-def:test"
            minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="test"
        type="oval-def:TestType" abstract="true">
        <xsd:annotation>
          <xsd:documentation>The test element is an
            abstract element that is meant to be
            extended (via substitution groups) by the
            individual tests found in the component
            schemas. An OVAL Test is used to compare
            an object(s) against a defined state. 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.</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:complexType name="TestType">
        <xsd:annotation>
          <xsd:documentation>The base type of every
            test includes an optional notes element
            and several 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 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.</xsd:documentation>
          <xsd:documentation>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.</xsd:documentation>
          <xsd:documentation>The optional
            check_existence attribute specifies how
            many items in the set defined by the OVAL
            Object must exist for the test to evaluate
            to true. The default value for this
            attribute is 'at_least_one_exists'
            indicating that by default the test may
            evaluate to true if at least one item
            defined by the OVAL Object exists on the
```



```
            system. For example, if a value of
            'all_exist' is given, every item defined
            by the OVAL Object must exist on the
            system for the test to evaluate to true.
            If the OVAL Object uses a variable
            reference, then every value of that
            variable must exist. Note that a pattern
            match defines a unique set of matching
            items found on a system. So when
            check_existence = 'all_exist' and a regex
            matches anything on a system the test will
            evaluate to true (since all matching
            objects on the system were found on the
            system). When check_existence =
            'all_exist' and a regex does not match
            anything on a system the test will
            evaluate to false.</xsd:documentation>
          <xsd:documentation>The required check
            attribute specifies how many items in the
            set defined by the OVAL Object (ignoring
            items with a status of Does Not Exist)
            must satisfy the state requirements. For
            example, should the test check that all
            matching files have a specified version or
            that at least one file has the specified
            version? The valid check values are
            explained in the description of the
            CheckEnumeration simple type. Note that if
            the test does not contain any references
            to OVAL States, then the check attribute
            has no meaning and can be ignored during
            evaluation.</xsd:documentation>
          <xsd:documentation>An OVAL Test evaluates to
            true if both the check_existence and check
            attributes are satisfied during
            evaluation. The evaluation result for a
            test is determined by first evaluating the
            check_existence attribute. If the result
            of evaluating the check_existence
            attribute is true then the check attribute
            is evaluated. An interpreter may choose to
            always evaluate both the check_existence
            and the check attributes, but once the
            check_existence attribute evaluation has
            resulted in false the overall test result
            after evaluating the check attribute will
            not be affected.</xsd:documentation>
          <xsd:documentation>The optional
```

```
          state_operator attribute provides the
          logical operator that combines the
          evaluation results from each referenced
          state on a per item basis. Each matching
          item is compared to each referenced state.
          The result of comparing each state to a
          single item is combined based on the
          specified state_operator value to
          determine one result for each item.
          Finally, the results for each item are
          combined based on the specified check
          value. Note that if the test does not
          contain any references to OVAL States,
          then the state_operator attribute has no
          meaning and can be ignored during
          evaluation. Referencing multiple states in
          one test allows ranges of possible values
          to be expressed. For example, one state
          can check that a value greater than 8 is
          found and another state can check that a
          value of less than 16 is found. In this
          example the referenced states are combined
          with a state_operator = 'AND' indicating
          that the conditions of all referenced
          states must be satisfied and that the
          value must be between 8 AND 16. The valid
          state_operation values are explained in
          the description of the OperatorEnumeration
          simple type.</xsd:documentation>
        <xsd:appinfo>
          <sch:pattern id="oval-def_test_type">
            <sch:rule
              context="oval-def:oval_definitions/
              oval-def:tests/*[@check_existence='none_exist']">
              <sch:assert
                test="not(*[local-name()='state'])"
                  ><sch:value-of select="@id"/> - No
                state should be referenced when
                check_existence has a value of
                'none_exist'.</sch:assert>
            </sch:rule>
          </sch:pattern>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element ref="ds:Signature"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="oval:notes" minOccurs="0"
```



```
          maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="id"
        type="oval:TestIDPattern" use="required"/>
      <xsd:attribute name="version"
        type="xsd:nonNegativeInteger" use="required"/>
      <xsd:attribute name="check_existence"
        type="oval:ExistenceEnumeration"
        use="optional" default="at_least_one_exists"/>
      <xsd:attribute name="check"
        type="oval:CheckEnumeration" use="required"/>
      <xsd:attribute name="state_operator"
        type="oval:OperatorEnumeration"
        use="optional" default="AND"/>
      <xsd:attribute name="comment"
        type="oval:NonEmptyStringType"
        use="required"/>
      <xsd:attribute name="deprecated"
        type="xsd:boolean" use="optional"
        default="false"/>
    </xsd:complexType>
```

```
    <xsd:complexType name="ObjectRefType">
      <xsd:annotation>
        <xsd:documentation>The ObjectRefType complex
          type defines an object reference to be
          used by OVAL Tests that are defined in the
          component schemas. The required object_ref
          attribute specifies the id of the OVAL
          Object being
          referenced.</xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="object_ref"
        type="oval:ObjectIDPattern" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="StateRefType">
      <xsd:annotation>
        <xsd:documentation>The StateRefType complex
          type defines a state reference to be used
          by OVAL Tests that are defined in the
          component schemas. The required state_ref
          attribute specifies the id of the OVAL
          State being
          referenced.</xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="state_ref"
        type="oval:StateIDPattern" use="required"/>
    </xsd:complexType>
  <!-- ================================================== -->
```



```
  <!-- ====================  OBJECTS  ======================== -->
  <!-- ================================================== -->
    <xsd:complexType name="ObjectsType">
      <xsd:annotation>
        <xsd:documentation>The ObjectsType complex
          type is a container for one or more object
          child elements. Each object element
          provides details that define a unique set
          of matching items to be used by an OVAL
          Test. Please refer to the description of
          the object element for more information
          about an individual
          object.</xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element ref="oval-def:object"
          minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="object"
      type="oval-def:ObjectType" abstract="true">
      <xsd:annotation>
        <xsd:documentation>The object element is an
          abstract element that is meant to be
          extended (via substitution groups) by the
          objects found in the component schemas. An
          actual object element is not valid. The
          use of this abstract element simplifies
          the OVAL schema by allowing individual
          objects to inherit any common elements and
          attributes from the base ObjectType.
          Please refer to the description of the
          ObjectType complex type for more
          information.</xsd:documentation>
        <xsd:documentation>An object is used to
          identify a set of items to collect. The
          author of a schema object must define
          sufficient object entities to allow a user
          to identify a unique item to be
          collected.</xsd:documentation>
        <xsd:documentation>A simple object typically
          results in a single file, process, etc
          being identified. But through the use of
          pattern matches, sets, and variables,
          multiple matching items can be identified.
          The set of items matching the object can
```

```
            then be used by an OVAL test and compared
            against an OVAL state.</xsd:documentation>
```



```
        </xsd:annotation>
      </xsd:element>
      <xsd:complexType name="ObjectType">
        <xsd:annotation>
          <xsd:documentation>The base type of every
            object includes an optional notes element.
            The notes element of an object should be
            used to hold 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.</xsd:documentation>
          <xsd:documentation>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.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element ref="ds:Signature"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element ref="oval:notes" minOccurs="0"
            maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id"
          type="oval:ObjectIDPattern" use="required"/>
        <xsd:attribute name="version"
          type="xsd:nonNegativeInteger" use="required"/>
        <xsd:attribute name="comment"
          type="oval:NonEmptyStringType"
          use="optional"/>
        <xsd:attribute name="deprecated"
          type="xsd:boolean" use="optional"
          default="false"/>
```



```
      </xsd:complexType>
      <xsd:element name="set">
        <xsd:annotation>
          <xsd:documentation>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
            unique set of items. This set of items is
            defined by one or two references to OVAL
            Objects that provide the criteria needed
            to collect a set of system items. These
            items can have one or more filters applied
            to allow a subset of those items to be
```

```
            specifically included or excluded from the
            overall set of items.</xsd:documentation>
          <xsd:documentation>The set element's
            object_reference refers to an existing
            OVAL Object. The set element's filter
            element provides a reference to an
            existing OVAL State and includes an
            optional action attribute. The filter's
            action attribute allows the author to
            specify whether matching items should be
            included or excluded from the overall set.
            The default filter action is to exclude
            all matching items. In other words, the
            filter can be thought of filtering items
            out by default.</xsd:documentation>
          <xsd:documentation>Each filter is applied to
            the items identified by each OVAL Object
            before the set_operator is applied. For
            example, if an object_reference points to
            an OVAL Object that identifies 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 applied to the set of items
            identified by the OVAL Object. Care must
            be taken to ensure that conflicting
            filters are not applied. It is possible to
            exclude all items with a size of 10 KB and
            then include only items with a size of 10
            KB. This example would result in the empty
            set.</xsd:documentation>
          <xsd:documentation>The required set_operator
```



```
            attribute defines how different child sets
            are combined to form the overall unique
            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.</xsd:documentation>
          <xsd:appinfo>
            <sch:pattern id="oval-def_setobjref">
              <sch:rule
                context="oval-def:oval_definitions/
                oval-def:objects/*/oval-def:set/
                oval-def:object_reference">
                <sch:assert
                  test="name(./../..) =
                  name(ancestor::oval-def:oval_definitions/
                  oval-def:objects/*[@id=current()])"
                    ><sch:value-of select="../../@id"
                  /> - Each object referenced by the
                  set must be of the same type as
                  parent object</sch:assert>
              </sch:rule>
              <sch:rule
                context="oval-def:oval_definitions/
                oval-def:objects/*/
                oval-def:set/oval-def:set/
                oval-def:object_reference">
                <sch:assert
                  test="name(./../../..) =
                  name(ancestor::oval-def:oval_definitions/
                  oval-def:objects/*[@id=current()])"
                    ><sch:value-of
                    select="../../../@id"/> - Each
                  object referenced by the set must be
                  of the same type as parent
                  object</sch:assert>
              </sch:rule>
              <sch:rule
                context="oval-def:oval_definitions/
```

```
                oval-def:objects/*/oval-def:set/
                oval-def:set/oval-def:set/
                oval-def:object_reference">
                <sch:assert
                  test="name(./../../../..) =
                  name(ancestor::oval-def:oval_definitions/
                  oval-def:objects/*[@id=current()])"
                    ><sch:value-of
```



```
                    select="../../../../@id"/> - Each
                  object referenced by the set must be
                  of the same type as parent
                  object</sch:assert>
              </sch:rule>
            </sch:pattern>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexType>
          <xsd:choice>
            <xsd:sequence>
              <xsd:element ref="oval-def:set"
                minOccurs="1" maxOccurs="2"/>
            </xsd:sequence>
            <xsd:sequence>
              <xsd:element name="object_reference"
                type="oval:ObjectIDPattern"
                minOccurs="1" maxOccurs="2"/>
              <xsd:element ref="oval-def:filter"
                minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:choice>
          <xsd:attribute name="set_operator"
            type="oval-def:SetOperatorEnumeration"
            use="optional" default="UNION"/>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="filter">
        <xsd:annotation>
          <xsd:documentation>The filter element
            provides a reference to an existing OVAL
            State and includes an optional action
            attribute. The action attribute is used to
            specify whether items that match the
            referenced OVAL State will be included in
            the resulting set or excluded from the
            resulting set.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="oval:StateIDPattern">
              <xsd:attribute name="action"
                type="oval-def:FilterActionEnumeration"
                use="optional" default="exclude"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
```



```
    <!-- ================================================== -->
    <!-- ====================  STATES  ==================== -->
    <!-- ================================================== -->
      <xsd:complexType name="StatesType">
        <xsd:annotation>
          <xsd:documentation>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.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element ref="oval-def:state"
            minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="state"
        type="oval-def:StateType" abstract="true">
        <xsd:annotation>
          <xsd:documentation>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.</xsd:documentation>
          <xsd:documentation>An OVAL State is a
            collection of one or more characteristics
            pertaining to a specific object type. The
            OVAL State is used by an OVAL Test to
            determine if a unique set of items
            identified on a system meet certain
            characteristics.</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:complexType name="StateType">
        <xsd:annotation>
          <xsd:documentation>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.</xsd:documentation>
          <xsd:documentation>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.</xsd:documentation>
          <xsd:documentation>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.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element ref="ds:Signature"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element ref="oval:notes" minOccurs="0"
            maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id"
```



```
          type="oval:StateIDPattern" use="required"/>
        <xsd:attribute name="version"
          type="xsd:nonNegativeInteger" use="required"/>
        <xsd:attribute name="operator"
          type="oval:OperatorEnumeration"
          use="optional" default="AND"/>
        <xsd:attribute name="comment"
          type="oval:NonEmptyStringType"
          use="optional"/>
        <xsd:attribute name="deprecated"
          type="xsd:boolean" use="optional"
          default="false"/>
      </xsd:complexType>
    <!-- ======================================================== -->
    <!-- ====================  VARIABLES  ======================= -->
    <!-- ======================================================== -->
      <xsd:complexType name="VariablesType">
        <xsd:annotation>
          <xsd:documentation>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.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element ref="oval-def:variable"
            minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="variable"
        type="oval-def:VariableType" abstract="true">
        <xsd:annotation>
          <xsd:documentation>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.</xsd:documentation>
          <xsd:documentation>If a variable defines a
            collection 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 reported during OVAL
            analysis.</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:complexType name="VariableType">
        <xsd:annotation>
          <xsd:documentation>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 VariableIDPattern 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 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.</xsd:documentation>
          <xsd:documentation>The required datatype
            attribute specifies the type of value
```



```
            being defined. The set of values
            identified by a variable must comply with
            the specified datatype, otherwise an error
            should be reported. Please see the
            DatatypeEnumeration for details about each
            valid datatype. For example, if the
            datatype of the variable is specified as
            boolean then the value(s) returned by the
            component / function should be "true",
            "false", "1", or "0".</xsd:documentation>
          <xsd:documentation>Note that the 'record'
            datatype is not permitted on variables.
            The notes section of a variable should be
            used to hold information that might be
            helpful to someone examining the technical
            aspects of the variable. Please refer to
            the description of the NotesType complex
            type for more information about the notes
            element.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element ref="ds:Signature"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element ref="oval:notes" minOccurs="0"
            maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="id"
          type="oval:VariableIDPattern" use="required"/>
```

```
        <xsd:attribute name="version"
          type="xsd:nonNegativeInteger" use="required"/>
        <xsd:attribute name="datatype" use="required"
          type="oval:SimpleDatatypeEnumeration">
          <xsd:annotation>
            <xsd:documentation>Note that the 'record'
              datatype is not permitted on
              variables.</xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="comment"
          type="oval:NonEmptyStringType"
          use="required"/>
        <xsd:attribute name="deprecated"
          type="xsd:boolean" use="optional"
          default="false"/>
      </xsd:complexType>
      <xsd:element name="external_variable"
        substitutionGroup="oval-def:variable">
        <xsd:annotation>
```



```
          <xsd:documentation>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.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
          <xsd:complexContent>
            <xsd:extension
              base="oval-def:VariableType">
              <xsd:choice minOccurs="0"
                maxOccurs="unbounded">
                <xsd:element name="possible_value"
                  type="oval-def:PossibleValueType"/>
                <xsd:element
                  name="possible_restriction"
                  type="oval-def:PossibleRestrictionType"
                />
              </xsd:choice>
            </xsd:extension>
          </xsd:complexContent>
        </xsd:complexType>
```

```
    </xsd:element>
    <xsd:complexType name="PossibleValueType">
      <xsd:annotation>
        <xsd:documentation>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.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:extension base="xsd:anySimpleType">
          <xsd:attribute name="hint"
            type="xsd:string" use="required"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="PossibleRestrictionType">
      <xsd:annotation>
        <xsd:documentation>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's evaluation is based on
          the operator attribute. The operator
          attribute is set to AND by default. Other
          valid operation values are explained in
          the description of the OperatorEnumeration
          simple type. One can think of the
          possible_value and possible_restriction
          elements as an OR'd list of possible
          values, with the restriction elements as
          using the selected operation to evaluate
          its own 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.</xsd:documentation>
      </xsd:annotation>
```



```
      <xsd:choice>
        <xsd:element name="restriction"
          type="oval-def:RestrictionType"
          minOccurs="1" maxOccurs="unbounded"/>
      </xsd:choice>
      <xsd:attribute name="operator"
        type="oval:OperatorEnumeration"
        use="optional" default="AND"/>
      <xsd:attribute name="hint" type="xsd:string"
        use="required"/>
    </xsd:complexType>
    <xsd:complexType name="RestrictionType">
      <xsd:annotation>
        <xsd:documentation>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.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:extension base="xsd:anySimpleType">
          <xsd:attribute name="operation"
            type="oval:OperationEnumeration"
            use="required"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:element name="constant_variable"
      substitutionGroup="oval-def:variable">
      <xsd:annotation>
        <xsd:documentation>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 a collection of
          values to be used throughout the
          evaluation of the OVAL Definition File in
          which it has been defined. Constant
          variables cannot be over-ridden by an
          external source. The actual value of a
          constant variable is defined by the
          required value child element. A collection
```



```
          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.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension
            base="oval-def:VariableType">
            <xsd:sequence>
              <xsd:element name="value"
                type="oval-def:ValueType"
                minOccurs="1" maxOccurs="unbounded"
              />
            </xsd:sequence>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="ValueType">
      <xsd:annotation>
        <xsd:documentation>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 cannot be over-ridden by an external
          source.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:extension base="xsd:anySimpleType"/>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:element name="local_variable"
      substitutionGroup="oval-def:variable">
      <xsd:annotation>
        <xsd:documentation>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.
          Each local variable is defined by either a
          single component or a complex function,
          meaning that a value can be as simple as a
```

```
          literal string or as complex as multiple
```



```
          registry keys concatenated together. Note
          that if an individual component is used
          and it returns a collection of values,
          then there will be multiple values
          associated with the local_variable. For
          example, if an object_component is used
          and it references a file object that
          identifies a set of 5 files, then the
          local variable would evaluate to a
          collection of those 5 values. Please refer
          to the description of the ComponentGroup
          for more information.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension
            base="oval-def:VariableType">
            <xsd:sequence>
              <xsd:group
                ref="oval-def:ComponentGroup"/>
            </xsd:sequence>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:group name="ComponentGroup">
      <xsd:annotation>
        <xsd:documentation>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.</xsd:documentation>
      </xsd:annotation>
      <xsd:choice>
        <xsd:element name="object_component"
          type="oval-def:ObjectComponentType"/>
        <xsd:element name="variable_component"
          type="oval-def:VariableComponentType"/>
        <xsd:element name="literal_component"
```



```
          type="oval-def:LiteralComponentType"/>
        <xsd:group ref="oval-def:FunctionGroup"/>
      </xsd:choice>
    </xsd:group>
    <xsd:complexType name="LiteralComponentType">
      <xsd:annotation>
        <xsd:documentation>The LiteralComponentType
          complex type defines a literal value to be
          used as a component. The optional datatype
          attribute defines the type of data
          expected. The default datatype is
          'string'.</xsd:documentation>
        <xsd:appinfo>
          <sch:pattern
            id="oval-def_literal_component">
            <sch:rule
```

```
              context="oval-def:literal_component">
              <sch:assert
                test="not(@datatype='record')"
                  ><sch:value-of
                  select="ancestor::*/@id"/> - The
                'record' datatype is prohibited on
                variables.</sch:assert>
            </sch:rule>
  <!--
    <sch:rule context="oval-def:literal_component
    /*/*[not(@datatype)]">
    </sch:rule>
    <sch:rule context=
    "oval-def:literal_component[@datatype='binary']">
    <sch:assert test="matches(., '^[0-9a-fA-F]*$')">
    <sch:value-of select="../@id"/> - A value of
    '<sch:value-of select="."/>' for the
    <sch:value-of select="name()"/>
    entity is not valid given a datatype of binary.</sch:assert>
    </sch:rule>
    <sch:rule context=
    "oval-def:literal_component[@datatype='boolean']">
    <sch:assert test="matches(., '^true$|^false$|^1$|^0$')">
    <sch:value-of select="../@id"/>
    - A value of '<sch:value-of select="."/>' for the
    <sch:value-of select="name()"/>
    entity is not valid given a datatype of boolean.
    </sch:assert>
    </sch:rule>
    <sch:rule context=
    "oval-def:literal_component[@datatype='evr_string']">
    <sch:assert test="matches(., '^[^:\-]*:[^:\-]*-[^:\-]*$')">
```



```
    <sch:value-of select="../@id"/>
    - A value of '<sch:value-of select="."/>' for the
    <sch:value-of select="name()"/>
    entity is not valid given a datatype of evr_string.
    </sch:assert>
    </sch:rule>
    <sch:rule context=
    "oval-def:literal_component[@datatype='fileset_revision']">
    </sch:rule>
    <sch:rule context=
    "oval-def:literal_component[@datatype='float']">
    <sch:assert test=
    "matches(., '^[+\-]?[0-9]+([\.][0-9]+)?
    ([eE][+\-]?[0-9]+)?$|^NaN$|^INF$|^\-INF$')">
    <sch:value-of select="../@id"/> - A value of
    '<sch:value-of select="."/>' for the
    <sch:value-of select="name()"/>
    entity is not valid given a datatype of float.</sch:assert>
    </sch:rule>
    <sch:rule context=
    "oval-def:literal_component[@datatype='ios_version']">
    </sch:rule>
    <sch:rule context=
    "oval-def:literal_component[@datatype='int']">
    <sch:assert test=
    "matches(., '^[+\-]?[0-9]+$')">
    <sch:value-of select="../@id"/> - A value of
    '<sch:value-of select="."/>' for the
    <sch:value-of select="name()"/>
    entity is not valid given a datatype of int.
    </sch:assert>
    </sch:rule>
    <sch:rule context=
    "oval-def:literal_component[@datatype='string']">
    </sch:rule>
    <sch:rule context=
    "oval-def:literal_component[@datatype='version']">
    </sch:rule>
  -->
        </sch:pattern>
      </xsd:appinfo>
```

```
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:extension base="xsd:anySimpleType">
          <xsd:attribute name="datatype"
            type="oval:SimpleDatatypeEnumeration"
            use="optional" default="string"/>
        </xsd:extension>
```



```
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="ObjectComponentType">
      <xsd:annotation>
        <xsd:documentation>The ObjectComponentType
          complex type defines a specific value or
          set of values on the local system to
          obtain.</xsd:documentation>
        <xsd:documentation>The required object_ref
          attribute provides a reference to an
          existing OVAL Object declaration. The
          referenced OVAL Object specifies a set of
          OVAL Items to collect. Note that an OVAL
          Object might identify 0, 1, or many OVAL
          Items on a system. If no items are found
          on the system then an error should be
          reported when determining the value of an
          ObjectComponentType. If 1 or more OVAL
          Items are found then each OVAL Item will
          be considered and the ObjectComponentType
          may have one or more
          values.</xsd:documentation>
        <xsd:documentation>The required item_field
          attribute specifies the name of the entity
          whose value will be retrieved from each
          OVAL Item collected by the referenced OVAL
          Object. For example, if the object_ref
          references a win-def:file_object, the
          item_field may specify the 'version'
          entity as the field to use as the value of
          the ObjectComponentType. Note that an OVAL
          Item may have 0, 1, or many entities whose
          name matches the specified item_field
          value. If an entity is not found with a
          name that matches the value of the
          item_field an error should be reported
          when determining the value of an
          ObjectComponentType. If 1 or more matching
          entities are found in a single OVAL Item
          the value of the ObjectComponentType is
          the list of the values from each of the
          matching entities.</xsd:documentation>
        <xsd:documentation>The optional record_field
          attribute specifies the name of a field in
          a record entity in an OVAL Item. The
          record_field attribute allows the value of
          a specific field to be retrieved from an
          entity with a datatype of 'record'. If a
```



```
          field with a matching name attribute value
          is not found in the referenced OVAL Item
          entity an error should be reported when
          determining the value of the
          ObjectComponentType.</xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="object_ref"
        type="oval:ObjectIDPattern" use="required"/>
      <xsd:attribute name="item_field"
        type="oval:NonEmptyStringType"
```

```
        use="required"/>
      <xsd:attribute name="record_field"
        type="oval:NonEmptyStringType"
        use="optional"/>
    </xsd:complexType>
    <xsd:complexType name="VariableComponentType">
      <xsd:annotation>
        <xsd:documentation>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.</xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="var_ref"
        type="oval:VariableIDPattern" use="required"
      />
    </xsd:complexType>
    <xsd:group name="FunctionGroup">
      <xsd:annotation>
        <xsd:documentation>Complex functions have
          been defined that help determine how to
          manipulate 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 reporting 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. It is also important to note
          that when using these functions with
          sub-components that return a collection of
          values that the operation will be
          performed on the Cartesian product of the
          components and the result is also a
          collection of values. For example, assume
          a local_variable specifies the arithmetic
          function with an arithmetic_operation of
          "add" and has two sub-components under
          this function: the first component returns
          "1" and "2", and the second component
          returns "3" and "4" and "5". The
          local_variable element would be evaluated
          to have a collection of six values: 1+3,
          1+4, 1+5, 2+3, 2+4, and 2+5. Please refer
          to the description of a specific function
          for more details about
          it.</xsd:documentation>
      </xsd:annotation>
      <xsd:choice>
        <xsd:element name="arithmetic"
          type="oval-def:ArithmeticFunctionType"/>
        <xsd:element name="begin"
          type="oval-def:BeginFunctionType"/>
        <xsd:element name="concat"
          type="oval-def:ConcatFunctionType"/>
        <xsd:element name="end"
```

```
          type="oval-def:EndFunctionType"/>
        <xsd:element name="escape_regex"
          type="oval-def:EscapeRegexFunctionType"/>
        <xsd:element name="split"
          type="oval-def:SplitFunctionType"/>
        <xsd:element name="substring"
          type="oval-def:SubstringFunctionType"/>
        <xsd:element name="time_difference"
          type="oval-def:TimeDifferenceFunctionType"/>
        <xsd:element name="regex_capture"
          type="oval-def:RegexCaptureFunctionType"/>
        <xsd:element name="unique"
          type="oval-def:UniqueFunctionType"/>
```



```
        <xsd:element name="count"
          type="oval-def:CountFunctionType"/>
        <xsd:element name="glob_to_regex"
          type="oval-def:GlobToRegexFunctionType"/>
      </xsd:choice>
    </xsd:group>
    <xsd:complexType name="ArithmeticFunctionType">
      <xsd:annotation>
        <xsd:documentation>The arithmetic function
          takes two or more integer or float
          components and performs a basic
          mathematical function on them. The result
          of this function is a single integer or
          float unless one of the components returns
          a collection of values. In this case the
          specified arithmetic function would be
          performed multiple times and the end
          result would also be a collection of
          values for the local variable. For example
          assume a local_variable specifies the
          arithmetic function with an
          arithmetic_operation of "add" and has two
          sub-components under this function: the
          first component returns "1" and "2", and
          the second component returns "3" and "4"
          and "5". The local_variable element would
          be evaluated to be a collection of six
          values: 1+3, 1+4, 1+5, 2+3, 2+4, and
          2+5.</xsd:documentation>
        <xsd:documentation>Note that if both an
          integer and float components are used then
          the result is a float.</xsd:documentation>
        <xsd:appinfo>
          <sch:pattern
            id="oval-def_arithmeticfunctionrules">
            <sch:rule
              context="oval-def:arithmetic/
              oval-def:literal_component">
              <sch:assert
                test="@datatype='float' or @datatype='int'"
                >A literal_component used by an
                arithmetic function must have a
                datatype of float or
                int.</sch:assert>
            </sch:rule>
            <sch:rule
              context="oval-def:arithmetic/
              oval-def:variable_component">
```



```
              <sch:let name="var_ref"
                value="@var_ref"/>
              <sch:assert
                test="ancestor::oval-def:oval_definitions/
```

```
                oval-def:variables/*[@id=$var_ref]/
                @datatype='float' or
                ancestor::oval-def:oval_definitions/
                oval-def:variables/*[@id=$var_ref]/
                @datatype='int'"
                >The variable referenced by the
                arithmetic function must have a
                datatype of float or
                int.</sch:assert>
              </sch:rule>
            </sch:pattern>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence minOccurs="2"
          maxOccurs="unbounded">
          <xsd:group ref="oval-def:ComponentGroup"/>
        </xsd:sequence>
        <xsd:attribute name="arithmetic_operation"
          type="oval-def:ArithmeticEnumeration"
          use="required"/>
      </xsd:complexType>
      <xsd:complexType name="BeginFunctionType">
        <xsd:annotation>
          <xsd:documentation>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 does not already start
            with the specified character (or string).
            If the component string does not start
            with the specified character (or string)
            the entire character (or string) will be
            prepended to the component
            string..</xsd:documentation>
          <xsd:appinfo>
            <sch:pattern
              id="oval-def_beginfunctionrules">
              <sch:rule
                context="oval-def:begin/
                oval-def:literal_component">
                <sch:assert
```



```
                test="not(@datatype) or @datatype='string'"
                >A literal_component used by the
                begin function must have a datatype
                of string.</sch:assert>
              </sch:rule>
              <sch:rule
                context="oval-def:begin/
                oval-def:variable_component">
                <sch:let name="var_ref"
                  value="@var_ref"/>
                <sch:assert
                  test="ancestor::oval-def:oval_definitions/
                  oval-def:variables/*[@id=$var_ref]/
                  @datatype = 'string'"
                  >The variable referenced by the
                  begin function must have a datatype
                  of string.</sch:assert>
              </sch:rule>
            </sch:pattern>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:group ref="oval-def:ComponentGroup"/>
        </xsd:sequence>
        <xsd:attribute name="character"
          type="xsd:string" use="required"/>
      </xsd:complexType>
      <xsd:complexType name="ConcatFunctionType">
        <xsd:annotation>
```

```
          <xsd:documentation>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 a collection 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 evaluate to a
            collection of 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.</xsd:documentation>
          <xsd:appinfo>
            <evaluation_documentation>Below is a chart
              that specifies how to classify the flag
              status of a variable using the concat
              function during evaluation when multiple
              components are supplied. Both the object
              and variable component are indirectly
              associated with collected objects in a
              system characteristics file. These
              objects could have been completely
              collected from the system, or there
              might have been some type of error that
              led to the object not being collected,
              or maybe only a part of the object set
              was collected. This flag status is
              important as OVAL Objects or OVAL States
              that are working with a variable
              (through the var_ref attribute on an
              entity) can use this information to
              report more accurate results. For
              example, an OVAL Test with a check
              attribute of 'at least one' that
              specifies an object with a variable
              reference, might be able to produce a
              valid result based on an incomplete
              object set as long as one of the objects
              in the set is
              true.</evaluation_documentation>
            <evaluation_chart xml:space="preserve">
        ||  num of components with flag      ||
        ||                                   || resulting flag is
        || E  | C  | I  | DNE | NC | NA      ||
   -----||-----------------------------------||------------------
        || 1+ | 0+ | 0+ | 0+  | 0+ | 0+      || Error
        || 0  | 1+ | 0  | 0   | 0  | 0       || Complete
        || 0  | 0+ | 1+ | 0   | 0  | 0       || Incomplete
        || 0  | 0+ | 0+ | 1+  | 0  | 0       || Does Not Exist
        || 0  | 0+ | 0+ | 0+  | 1+ | 0       || Not Collected
        || 0  | 0+ | 0+ | 0+  | 0+ | 1+      || Not Applicable
   -----||-----------------------------------||------------------
                  </evaluation_chart>
            <sch:pattern
              id="oval-def_concatfunctionrules">
              <sch:rule
                context="oval-def:concat/
                oval-def:literal_component">
```

```
              <sch:assert
                test="not(@datatype) or @datatype='string'"
                >A literal_component used by the
                concat function must have a datatype
                of string.</sch:assert>
            </sch:rule>
            <sch:rule
              context="oval-def:concat/
              oval-def:variable_component">
              <sch:let name="var_ref"
                value="@var_ref"/>
              <sch:assert
                test="ancestor::oval-def:oval_definitions/
                oval-def:variables/*[@id=$var_ref]/
                @datatype = 'string'"
                >The variable referenced by the
                concat function must have a datatype
                of string.</sch:assert>
            </sch:rule>
          </sch:pattern>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence minOccurs="2"
        maxOccurs="unbounded">
        <xsd:group ref="oval-def:ComponentGroup"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="EndFunctionType">
      <xsd:annotation>
        <xsd:documentation>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 does not already end with
          the specified character (or string). If
          the desired end character is a string,
          then the entire end string must exist at
          the end if the component string. If the
          entire end string is not present then the
          entire end string is appended to the
          component string.</xsd:documentation>
        <xsd:appinfo>
          <sch:pattern
            id="oval-def_endfunctionrules">
            <sch:rule
```



```
              context="oval-def:end/oval-def:literal_component">
              <sch:assert
                test="not(@datatype) or @datatype='string'"
                >A literal_component used by the end
                function must have a datatype of
                string.</sch:assert>
            </sch:rule>
            <sch:rule
              context="oval-def:end/oval-def:variable_component">
              <sch:let name="var_ref"
                value="@var_ref"/>
              <sch:assert
                test="ancestor::oval-def:oval_definitions/
                oval-def:variables/*[@id=$var_ref]/
                @datatype = 'string'"
                >The variable referenced by the end
                function must have a datatype of
                string.</sch:assert>
            </sch:rule>
          </sch:pattern>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
```

```
        <xsd:group ref="oval-def:ComponentGroup"/>
      </xsd:sequence>
      <xsd:attribute name="character"
        type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="EscapeRegexFunctionType">
      <xsd:annotation>
        <xsd:documentation>The escape_regex function
          takes a single string component and
          escapes all of the regular expression
          characters. If the string sub-component
          contains multiple values, then the
          escape_regex function will be applied to
          each individual value and return a
          multiple-valued result. For example, the
          string '(\.test_string*)?' will evaluate
          to '\(\\\.test_string\*\)\?'. The purpose
          for this is that many times, a component
          used in pattern match needs to be treated
          as a literal string and not a regular
          expression. For example, assume a basic
          component element that identifies a file
          path that is held in the Windows registry.
          This path is a string that might contain
          regular expression characters. These
```



```
          characters are likely not intended to be
          treated as regular expression characters
          and need to be escaped. This function
          allows a definition writer to mark convert
          the values of components to regular
          expression format.</xsd:documentation>
        <xsd:documentation>Note that when using
          regular expressions, OVAL supports a
          common subset of the regular expression
          character classes, operations, expressions
          and other lexical tokens defined within
          Perl 5's regular expression specification.
          The set of Perl metacharacters which must
          be escaped by this function is as follows,
          enclosed by single quotes:
          '^$\.[](){}*+?|'. For more information on
          the supported regular expression syntax in
          OVAL see:
          http://oval.mitre.org/language/
          about/re_support_5.6.html.</xsd:documentation>
        <xsd:appinfo>
          <sch:pattern
            id="oval-def_escaperegexfunctionrules">
            <sch:rule
              context="oval-def:escape_regex/
              oval-def:literal_component">
              <sch:assert
                test="not(@datatype) or @datatype='string'"
                >A literal_component used by the
                escape_regex function must have a
                datatype of string.</sch:assert>
            </sch:rule>
            <sch:rule
              context="oval-def:escape_regex/
              oval-def:variable_component">
              <sch:let name="var_ref"
                value="@var_ref"/>
              <sch:assert
                test="ancestor::oval-def:oval_definitions/
                oval-def:variables/*[@id=$var_ref]/
                @datatype = 'string'"
                >The variable referenced by the
                escape_regex function must have a
                datatype of string.</sch:assert>
            </sch:rule>
          </sch:pattern>
        </xsd:appinfo>
      </xsd:annotation>
```

```
      <xsd:sequence>
        <xsd:group ref="oval-def:ComponentGroup"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="SplitFunctionType">
      <xsd:annotation>
        <xsd:documentation>The split function takes
          a single string component and turns it
          into a collection of values based on a
          delimiter string. For example, assume that
          a basic component element returns the
          value "a-b-c-d" to the split function with
          the delimiter set to "-". The
          local_variable element would be evaluated
          to have four values "a", "b", "c", and
          "d". If the basic component returns a
          value that begins, or ends, with a
          delimiter, the local_variable element
          would contain empty string values at the
          beginning, or end, of the collection of
          values returned for that string component.
          For example, if the delimiter is "-", and
          the basic component element returns the
          value "-a-a-", the local_variable element
          would evaluate to a collection of four
          values "", "a", "a", and "". Likewise, if
          the basic component element returns a
          value that contains adjacent delimiters
          such as "---", the local_variable element
          would evaluate to a collection of four
          values "", "", "", and "". Lastly, if the
          basic component element used by the split
          function returnsa collection of values,
          then the split function is performed
          multiple times, and all of the results,
          from each of the split functions, are
          returned.</xsd:documentation>
        <xsd:appinfo>
          <sch:pattern
            id="oval-def_splitfunctionrules">
            <sch:rule
              context="oval-def:split/
              oval-def:literal_component">
              <sch:assert
                test="not(@datatype) or @datatype='string'"
                >A literal_component used by the
                split function must have a datatype
                of string.</sch:assert>
```



```
            </sch:rule>
            <sch:rule
              context="oval-def:split/
              oval-def:variable_component">
              <sch:let name="var_ref"
                value="@var_ref"/>
              <sch:assert
                test="ancestor::oval-def:oval_definitions/
                oval-def:variables/*[@id=$var_ref]/
                @datatype
                = 'string'"
                >The variable referenced by the
                split function must have a datatype
                of string.</sch:assert>
            </sch:rule>
          </sch:pattern>
        </xsd:appinfo>
```

```
      </xsd:annotation>
      <xsd:sequence>
        <xsd:group ref="oval-def:ComponentGroup"/>
      </xsd:sequence>
      <xsd:attribute name="delimiter"
        type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="SubstringFunctionType">
      <xsd:annotation>
        <xsd:documentation>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.
          To include the first character of the
          string, the start position would be 1. A
          value less than 1 also means that the
          start position would be 1. If the
          substring_start attribute has value
          greater than the length of the original
          string an error should be reported. The
          substring_length attribute defines how
          many characters after, and including, the
          starting character to include. A
          substring_length value greater than the
          actual length of the string, or a negative
          value, means to include all of 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 evaluate to
          have a single value of "cd". If the string
          component used by the substring function
          returns a collection of values, then the
          substring operation is performed multiple
          times and results in a collection of
          values for the
          component.</xsd:documentation>
        <xsd:appinfo>
          <sch:pattern
            id="oval-def_substringfunctionrules">
            <sch:rule
              context="oval-def:substring/
              oval-def:literal_component">
              <sch:assert
                test="not(@datatype) or @datatype='string'"
                >A literal_component used by the
                substring function must have a
                datatype of string.</sch:assert>
            </sch:rule>
            <sch:rule
              context="oval-def:substring/
              oval-def:variable_component">
              <sch:let name="var_ref"
                value="@var_ref"/>
              <sch:assert
                test="ancestor::oval-def:oval_definitions/
                oval-def:variables/*[@id=$var_ref]/
                @datatype = 'string'"
                >The variable referenced by the
                substring function must have a
                datatype of string.</sch:assert>
            </sch:rule>
          </sch:pattern>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:group ref="oval-def:ComponentGroup"/>
      </xsd:sequence>
      <xsd:attribute name="substring_start"
```

```
        type="xsd:int" use="required"/>
      <xsd:attribute name="substring_length"
        type="xsd:int" use="required"/>
    </xsd:complexType>
    <xsd:complexType
      name="TimeDifferenceFunctionType">
```



```
      <xsd:annotation>
        <xsd:documentation>The time_difference
          function calculates the difference in
          seconds between date-time values. If one
          component is specified, the values of that
          component are subtracted from the current
          time (UTC). The current time is the time
          at which the function is evaluated. If two
          components are specified, the value of the
          second component is subtracted from the
          value of the first component. If the
          component(s) contain a collection of
          values, the operation is performed
          multiple times on the Cartesian product of
          the component(s) and the result is also a
          collection of time difference values. For
          example, assume a local_variable specifies
          the time_difference function and has two
          sub-components under this function: the
          first component returns "04/02/2009" and
          "04/03/2009", and the second component
          returns "02/02/2005" and "02/03/2005" and
          "02/04/2005". The local_variable element
          would evaluate to a collection of six
          values: (ToSeconds("04/02/2009") -
          ToSeconds("02/02/2005")),
          (ToSeconds("04/02/2009") -
          ToSeconds("02/03/2005")),
          (ToSeconds("04/02/2009") -
          ToSeconds("02/04/2005")),
          (ToSeconds("04/03/2009") -
          ToSeconds("02/02/2005")),
          (ToSeconds("04/03/2009") -
          ToSeconds("02/03/2005")), and
          (ToSeconds("04/03/2009") -
          ToSeconds("02/04/2005")).</xsd:documentation>
        <xsd:documentation>The date-time format of
          each component is determined by the two
          format attributes. The format1 attribute
          applies to the first component, and the
          format2 attribute applies to the second
          component. Valid values for the attributes
          are 'win_filetime', 'seconds_since_epoch',
          'day_month_year', 'year_month_day', and
          'month_day_year'. Please see the
          DateTimeFormatEnumeration for more
          information about each of these values. If
          an input value is not understood, the
```



```
          result is an error. If only one input is
          specified, specify the format with the
          format2 attribute, as the first input is
          considered to be the implied 'current
          time' input.</xsd:documentation>
        <xsd:documentation>Note that the datatype
          associated with the components should be
          'string' or 'int' depending on which date
          time format is specified. The result of
          this function though is always an
          integer.</xsd:documentation>
```

```
          <xsd:appinfo>
            <sch:pattern
              id="oval-def_timedifferencefunctionrules">
              <sch:rule
                context="oval-def:time_difference/
                oval-def:literal_component">
                <sch:assert
                  test="not(@datatype) or
                  @datatype='string' or
                  @datatype='int'"
                  >A literal_component used by the
                  time_difference function must have a
                  datatype of string or
                  int.</sch:assert>
              </sch:rule>
              <sch:rule
                context="oval-def:time_difference/
                oval-def:variable_component">
                <sch:let name="var_ref"
                  value="@var_ref"/>
                <sch:assert
                  test="ancestor::oval-def:oval_definitions/
                  oval-def:variables/*[@id=$var_ref]/
                  @datatype='string' or
                  ancestor::oval-def:oval_definitions/
                  oval-def:variables/*[@id=$var_ref]/
                  @datatype='int'"
                  >The variable referenced by the
                  time_difference function must have a
                  datatype of string or
                  int.</sch:assert>
              </sch:rule>
            </sch:pattern>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence minOccurs="1" maxOccurs="2">
          <xsd:group ref="oval-def:ComponentGroup"/>
```



```
        </xsd:sequence>
        <xsd:attribute name="format_1"
          type="oval-def:DateTimeFormatEnumeration"
          use="optional" default="year_month_day"/>
        <xsd:attribute name="format_2"
          type="oval-def:DateTimeFormatEnumeration"
          use="optional" default="year_month_day"/>
      </xsd:complexType>
      <xsd:complexType name="RegexCaptureFunctionType">
        <xsd:annotation>
          <xsd:documentation>The regex_capture
            function captures a single substring from
            a single string component. If the string
            sub-component contains multiple values,
            then the regex_capture function will
            extract a substring from each value. The
            'pattern' attribute provides a regular
            expression that should contain a single
            subexpression (using parentheses). For
            example, the pattern ^abc(.*)xyz$ would
            capture a substring from each of the
            string component's values if the value
            starts with abc and ends with xyz. In this
            case the subexpression would be all the
            characters that exist in between the abc
            and the xyz. Note that subexpressions
            match the longest possible
            substrings.</xsd:documentation>
          <xsd:documentation>If the regular expression
            contains multiple capturing sub-patterns,
            only the first capture is used. If there
            are no capturing sub-patterns, the result
            for each target string must be the empty
            string. Otherwise, if the regular
            expression could match the target string
            in more than one place, only the first
```

```
            match (and its first capture) is used. If
            no matches are found in a target string,
            the result for that target must be the
            empty string.</xsd:documentation>
          <xsd:documentation>Note that a quantified
            capturing sub-pattern does not produce
            multiple substrings. Standard regular
            expression semantics are such that if a
            capturing sub-pattern is required to match
            multiple times in order for the overall
            regular expression to match, the capture
            produced is the last substring to have
```



```
            matched the
            sub-pattern.</xsd:documentation>
          <xsd:documentation>Note that when using
            regular expressions, OVAL supports a
            common subset of the regular expression
            character classes, operations, expressions
            and other lexical tokens defined within
            Perl 5's regular expression specification.
            If any of the Perl metacharacters are to
            be used literally, then they must be
            escaped. The set of metacharacters which
            must be escaped for this purpose is as
            follows, enclosed by single quotes:
            '^$\.[](){}*+?|'. For more information on
            the supported regular expression syntax in
            OVAL see:
            http://oval.mitre.org/language/
            about/re_support_5.6.html.</xsd:documentation>
          <xsd:appinfo>
            <sch:pattern
              id="oval-def_regexcapturefunctionrules">
              <sch:rule
                context="oval-def:regex_capture/
                oval-def:literal_component">
                <sch:assert
                  test="not(@datatype) or @datatype='string'"
                  >A literal_component used by the
                  regex_capture function must have a
                  datatype of string.</sch:assert>
              </sch:rule>
              <sch:rule
                context="oval-def:regex_capture/
                oval-def:variable_component">
                <sch:let name="var_ref"
                  value="@var_ref"/>
                <sch:assert
                  test="ancestor::oval-def:oval_definitions/
                  oval-def:variables/*[@id=$var_ref]/
                  @datatype = 'string'"
                  >The variable referenced by the
                  regex_capture function must have a
                  datatype of string.</sch:assert>
              </sch:rule>
            </sch:pattern>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:group ref="oval-def:ComponentGroup"/>
```



```
        </xsd:sequence>
        <xsd:attribute name="pattern"
          type="xsd:string"/>
      </xsd:complexType>
      <xsd:complexType name="UniqueFunctionType">
```

```
      <xsd:annotation>
        <xsd:documentation>The unique function takes
          one or more components and removes any
          duplicate value from the set of
          components. All components used in the
          unique function will be treated as
          strings. For example, assume that three
          components exist, one that contains a
          string value of 'foo', and two of which
          both resolve to the string value 'bar'.
          Applying the unique function to these
          three components resolves to a
          local_variable with two string values,
          'foo' and 'bar'. Additionally, if any of
          the components referenced by the unique
          function evaluate to a collection of
          values, then those values are used in the
          unique calculation. For example, assume
          that there are two components, one of
          which resolves to a single string value,
          'foo', the other of which resolves to two
          string values, 'foo' and 'bar'. If the
          unique function is used to remove
          duplicates from these two components, the
          function will resolve to a local_variable
          that is a collection of two string values,
          'foo' and 'bar'.</xsd:documentation>
      </xsd:annotation>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:group ref="oval-def:ComponentGroup"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CountFunctionType">
      <xsd:annotation>
        <xsd:documentation>The count function takes
          one or more components and returns the
          count of all of the values represented by
          the components. For example, assume that
          two variables exist, each with a single
          value. By applying the count function
          against two variable components that
          resolve to the two variables, the
          resulting local_variable would have a
```



```
          value of '2'. Additionally, if any of the
          components referenced by the count
          function evaluate to a collection of
          values, then those values are used in the
          count calculation. For example, assume
          that there are two components, one of
          which resolves to a single value, the
          other of which resolves to two values. If
          the count function is used to provide a
          count of these two components, the
          function will resolve to a local_variable
          with the values '3'.</xsd:documentation>
      </xsd:annotation>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:group ref="oval-def:ComponentGroup"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="GlobToRegexFunctionType">
      <xsd:annotation>
        <xsd:documentation> The glob_to_regex
          function takes a single string component
          representing shell glob pattern and
          produces a single value that corresponds
          to result of a conversion of the original
          glob pattern into Perl 5's regular
          expression pattern. The glob_noescape
          attribute defines the way how the
          backslash ('\') character should be
          interpreted. It defaults to 'false'
          meaning backslash should be interpreted as
```

```
            an escape character (backslash is allowed
            to be used as an escape character). If the
            glob_noescape attribute would be set to
            'true' it instructs the glob_to_regex
            function to interpret the backslash ('\')
            character as a literal, rather than as an
            escape character (backslash is *not*
            allowed to be used as an escape
            character). Refer to table with examples
            below to see the difference how a
            different boolean value of the
            'glob_noescape' attribute will impact the
            output form of the resulting Perl 5's
            regular expression produced by
            glob_to_regex
            function.</xsd:documentation>
          <xsd:documentation>Please note the
            glob_to_regex function will fail to
```



```
            perform the conversion and return an error
            when the provided string argument (to
            represent glob pattern) does not represent
            a syntactically correct glob pattern. For
            example given the 'a*b?[' as the argument
            to be converted, glob_to_regex would
            return an error since there's missing the
            corresponding closing bracket in the
            provided glob pattern
            argument.</xsd:documentation>
          <xsd:documentation>Also, it is necessary to
            mention that the glob_to_regex function
            respects the default behaviour for the
            input glob pattern and output Perl 5's
            regular expression spaces. Namely this
            means that:</xsd:documentation>
          <xsd:documentation> - glob_to_regex will
            respect the UNIX glob behavior when
            processing forward slashes, forward slash
            should be treated as a path separator and
            * or ? shall not match
            it,</xsd:documentation>
          <xsd:documentation> - glob_to_regex will
            rule out matches having special meaning
            (for example '.' as a representation of
            the current working directory or '..' as a
            representation of the parent directory of
            the current working
            directory,</xsd:documentation>
          <xsd:documentation> - glob_to_regex will
            rule out files or folders starting with
            '.' character (e.g. dotfiles) unless the
            respective glob pattern part itself starts
            with the '.'
            character,</xsd:documentation>
          <xsd:documentation> - glob_to_regex will not
            perform case-sensitivity transformation
            (alphabetical characters will be copied
            from input glob pattern space to output
            Perl 5's regular expression pattern space
            intact). It is kept as a responsibility of
            the OVAL content author to provide input
            glob pattern argument in such case so the
            resulting Perl 5's regular expression
            pattern will match the expected pathname
            entries according to the case of
            preference,</xsd:documentation>
          <xsd:documentation> - glob_to_regex will not
```

```
          perform any possible brace expansion.
          Therefore glob patterns like
          '{pat,pat,pat}' would be converted into
          Perl 5's regular expression syntax in the
          original un-expanded form (kept for any
          potential subsequent expansion to be
          performed by Perl 5's regular expression
          engine in the moment of the use of that
          resulting regular
          expression),</xsd:documentation>
        <xsd:documentation> - glob_to_regex will not
          perform tilde ('~') character substitution
          to user name home directory pathname. The
          ('~') character will be passed to Perl 5's
          regular expression engine intact. If user
          name home directory pathname glob pattern
          behaviour is expected, the pathname of the
          user name home directory needs to be
          specified in the original input glob
          pattern already,</xsd:documentation>
        <xsd:documentation> - glob_to_regex function
          will not perform any custom changes wrt to
          the ordering of items (perform any
          additional sorting of set of pathnames
          represented by the provided glob pattern
          argument).</xsd:documentation>
        <xsd:appinfo>
          <evaluation_documentation>Below are some
            examples that outline how the
            glob_noescape attribute value affects
            the output form of the produced Perl
            regular expression. The far left column
            identifies the shell glob pattern
            provided as the input string component
            to the glob_to_regex function. The
            middle column specifies the two possible
            different boolean values of the
            'glob_noescape' attribute that can be
            used. Finally the last column depicts
            how the output produced by the
            glob_to_regex function - the resulting
            Perl regular expression would look
            like.</evaluation_documentation>
          <evaluation_chart xml:space="preserve">
                      ||             ||
            input     || glob_       || corresponding
            shell     || noescape    ||
            glob      || attribute   || Perl
```



```
              pattern || value     || Regular
                      ||           || Expression
    ------------------||-----------||-------------------------------
     '\*'             || false     || ^\*$
                      ||-----------||-------------------------------
     '\*'             || true      || ^\\[^/]*$
    ------------------||-----------||-------------------------------
     '\?'             || false     || ^\?$
                      ||-----------||-------------------------------
     '\?'             || true      || ^\\[^./]$
    ------------------||-----------||-------------------------------
     '\[hello\]'      || false     || ^\[hello\]$
                      ||-----------||-------------------------------
     '\[hello\]'      || true      || ^\\[hello\\]$
    ------------------||-----------||-------------------------------
     '/root/*'        || false     || ^/root/(?=[^.])[^/]*$
                      ||-----------||-------------------------------
     '/root/.*'       || false     || ^/root/\.[^/]*$
                      ||-----------||-------------------------------
     '/root/x*'       || false     || ^/root/x[^/]*$
                      ||-----------||-------------------------------
     '/root/?'        || false     || ^/root/[^./]$
                      ||-----------||-------------------------------
     '/root/.?'       || false     || ^/root/\.[^/]$
```

'/root/x?'	false	^/root/x[^/]$
'list.?'	false	^list\.[^/]$
'list.?'	true	^list\.[^/]$
'project.*'	false	^project\.[^/]*$
'project.*'	true	^project\.[^/]*$
'*old'	false	^(?=[^.])[^/]*old$
'*old'	true	^(?=[^.])[^/]*old$
'type*.[ch]'	false	^type[^/]*\.[ch]$
'type*.[ch]'	true	^type[^/]*\.[ch]$
'*.*'	false	^(?=[^.])[^/]*\.[^/]*$
'*.*'	true	^(?=[^.])[^/]*\.[^/]*$
'*'	false	^(?=[^.])[^/]*$

'*'	true	^(?=[^.])[^/]*$
'?'	false	^[^./]$
'?'	true	^[^./]$
'*'	false	^*$
'*'	true	^\\[^/]*$
'\?'	false	^\?$
'\?'	true	^\\[^./]$
'x[[:digit:]]*'	false	^x[[:digit:]]*$
'x[[:digit:]]*'	true	^x[[:digit:]]\\[^/]*$
''	false	^$
''	true	^$
'~/files/*.txt'	false	^~/files/(?=[^.])[^/]*\.txt$
'~/files/*.txt'	true	^~/files/(?=[^.])[^/]*\.txt$
'\'	false	^\\$
'\'	true	^\\$
'[ab'	false	INVALID
'[ab'	true	INVALID
'.*.conf'	false	^\.[^/]*\.conf$
'.*.conf'	true	^\.[^/]*\.conf$
'docs/?b'	false	^docs/[^./]b$
'docs/?b'	true	^docs/[^./]b$
'xy/??z'	false	^xy/[^./][^/]z$
'xy/??z'	true	^xy/[^./][^/]z$

</evaluation_chart>

```
          <sch:pattern
            id="oval-def_globtoregexfunctionrules">
            <sch:rule
              context="oval-def:glob_to_regex/
              oval-def:literal_component">
              <sch:assert
                test="not(@datatype) or @datatype='string'"
                >A literal_component used by the
                glob_to_regex function must have a
                datatype of string.</sch:assert>
            </sch:rule>
            <sch:rule
              context="oval-def:glob_to_regex/
              oval-def:variable_component">
              <sch:let name="var_ref"
                value="@var_ref"/>
              <sch:assert
                test="ancestor::oval-def:oval_definitions/
                oval-def:variables/*[@id=$var_ref]/
                @datatype = 'string'"
                >The variable referenced by the
                glob_to_regex function must have a
                datatype of string.</sch:assert>
            </sch:rule>
          </sch:pattern>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:group ref="oval-def:ComponentGroup"/>
      </xsd:sequence>
      <xsd:attribute name="glob_noescape"
        type="xsd:boolean" use="optional"
        default="false"/>
    </xsd:complexType>
  <!-- ======================================================== -->
  <!-- ====================  SIGNATURE  ======================== -->
  <!-- ======================================================== -->
    <!--
  The signature element is defined by the xmldsig schema.
  Please refer to that documentation for a description of the valid
  elements and types.  More information about the official W3C
  Recommendation regarding XML digital signatures can be found at
  http://www.w3.org/TR/xmldsig-core/.
        -->
  <!-- ======================================================== -->
  <!-- ==================  ENUMERATIONS  ====================== -->
  <!-- ======================================================== -->
    <xsd:simpleType name="ArithmeticEnumeration">
```



```
      <xsd:annotation>
        <xsd:documentation>The ArithmeticEnumeration
          simple type defines basic arithmetic
          operations. Currently add and multiply are
          defined.</xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="add"/>
        <xsd:enumeration value="multiply"/>
  <!--
      NOTE - we need to add a required position
      attribute to the components before we
      can have a subtract or divide function.
      This will have to wait for the next
      major release

      <xsd:enumeration value="divide"/>
      <xsd:enumeration value="subtract"/>
```

```
  -->
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="DateTimeFormatEnumeration">
      <xsd:annotation>
        <xsd:documentation>The
          DateTimeFormatEnumeration simple type
          defines the different date-time formats
          that are understood by OVAL. Note that in
          some cases there are a few different
          possibilities within a given format. Each
          of these possibilities is unique though
          and can be distinguished from each other.
          The different formats are used to clarify
          the higher level structure of the
          date-time string being
          used.</xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="year_month_day">
          <xsd:annotation>
            <xsd:documentation>The year_month_day
              value specifies date-time strings that
              follow the formats: 'yyyymmdd',
              'yyyymmddThhmmss', 'yyyy/mm/dd
              hh:mm:ss', 'yyyy/mm/dd', 'yyyy-mm-dd
              hh:mm:ss', or
              'yyyy-mm-dd'</xsd:documentation>
          </xsd:annotation>
        </xsd:enumeration>
```



```
        <xsd:enumeration value="month_day_year">
          <xsd:annotation>
            <xsd:documentation>The month_day_year
              value specifies date-time strings that
              follow the formats: 'mm/dd/yyyy
              hh:mm:ss', 'mm/dd/yyyy', 'mm-dd-yyyy
              hh:mm:ss', 'mm-dd-yyyy', 'NameOfMonth,
              dd yyyy hh:mm:ss' or 'NameOfMonth, dd
              yyyy', 'AbreviatedNameOfMonth, dd yyyy
              hh:mm:ss', or 'AbreviatedNameOfMonth,
              dd yyyy'</xsd:documentation>
          </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration value="day_month_year">
          <xsd:annotation>
            <xsd:documentation>The day_month_year
              value specifies date-time strings that
              follow the formats: 'dd/mm/yyyy
              hh:mm:ss', 'dd/mm/yyyy', 'dd-mm-yyyy
              hh:mm:ss', or
              'dd-mm-yyyy'</xsd:documentation>
          </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration value="win_filetime">
          <xsd:annotation>
            <xsd:documentation>The win_filetime
              value specifies date-time strings that
              follow the windows file time
              format.</xsd:documentation>
          </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration value="seconds_since_epoch">
          <xsd:annotation>
            <xsd:documentation>The
              seconds_since_epoch value specifies
              date-time values that represent the
              time in seconds since the UNIX epoch.
              The Unix epoch is the time 00:00:00
              UTC on January 1,
              1970.</xsd:documentation>
          </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration value="cim_datetime">
```

```
            <xsd:annotation>
              <xsd:documentation>The cim_datetime
                model is used by WMI and its value
                specifies date-time strings that
                follow the format:
```



```
                'yyyymmddHHMMSS.mmmmmmsUUU', and
                alternatively 'yyyy-mm-dd
                HH:MM:SS:mmm' only when used in WMI
                Query Language
                queries.</xsd:documentation>
            </xsd:annotation>
          </xsd:enumeration>
        </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType name="FilterActionEnumeration">
        <xsd:annotation>
          <xsd:documentation>The
            FilterActionEnumeration simple type
            defines the different options for
            filtering sets of
            items.</xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="exclude">
            <xsd:annotation>
              <xsd:documentation>The exclude value
                specifies that all items that match
                the filter shall be excluded from set
                that the filter is applied
                to.</xsd:documentation>
            </xsd:annotation>
          </xsd:enumeration>
          <xsd:enumeration value="include">
            <xsd:annotation>
              <xsd:documentation>The include value
                specifies that only items that match
                the filter shall be included in the
                set that the filter is applied
                to.</xsd:documentation>
            </xsd:annotation>
          </xsd:enumeration>
        </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType name="SetOperatorEnumeration">
        <xsd:annotation>
          <xsd:documentation>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 unique 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.</xsd:documentation>
          <xsd:appinfo>
            <evaluation_documentation>Below are some
              tables that outline how different flags
              are combined with a given set_operator
              to return a new flag. These tables are
              needed when computing the flag for
              collected objects that represent object
              sets in an OVAL Definition. The top row
              identifies the flag associated with the
```

```
          first set or object reference. The left
          column identifies the flag associated
          with the second set or object reference.
          The matrix inside the table represent
          the resulting flag when the given
          set_operator is applied. (E=error,
          C=complete, I=incomplete, DNE=does not
          exist, NC=not collected, NA=not
          applicable)</evaluation_documentation>
        <evaluation_chart xml:space="preserve">
                  ||                                    ||
 set_operator is  ||            obj 1 flag              ||
      union       ||                                    ||
                  ||  E  |  C  |  I  | DNE | NC  | NA  ||
 -----------------||-----------------------------------||
               E  ||  E  |  E  |  E  |  E  |  E  |  E  ||
  obj          C  ||  E  |  C  |  I  |  C  |  I  |  C  ||
   2           I  ||  E  |  I  |  I  |  I  |  I  |  I  ||
  flag       DNE  ||  E  |  C  |  I  | DNE |  I  | DNE ||
              NC  ||  E  |  I  |  I  |  I  |  NC |  NC ||
              NA  ||  E  |  C  |  I  | DNE |  NC |  NA ||
 -----------------||-----------------------------------||
                        </evaluation_chart>
        <evaluation_chart xml:space="preserve">
                  ||                                    ||
 set_operator is  ||             obj 1 flag             ||
  intersection    ||                                    ||
                  ||  E  |  C  |  I  | DNE | NC  | NA  ||
 -----------------||-----------------------------------||
               E  ||  E  |  E  |  E  | DNE |  E  |  E  ||
   obj         C  ||  E  |  C  |  I  | DNE |  NC |  C  ||
    2          I  ||  E  |  I  |  I  | DNE |  NC |  I  ||
   flag      DNE  || DNE | DNE | DNE | DNE | DNE | DNE ||
              NC  ||  E  |  NC |  NC | DNE |  NC |  NC ||
              NA  ||  E  |  C  |  I  | DNE |  NC |  NA ||
 -----------------||-----------------------------------||
```



```
                        </evaluation_chart>
        <evaluation_chart xml:space="preserve">
                  ||                                    ||
 set_operator is  ||             obj 1 flag             ||
    complement    ||                                    ||
                  ||  E  |  C  |  I  | DNE | NC  | NA  ||
 -----------------||-----------------------------------||
               E  ||  E  |  E  |  E  | DNE |  E  |  E  ||
   obj         C  ||  E  |  C  |  I  | DNE |  NC |  E  ||
    2          I  ||  E  |  E  |  E  | DNE |  NC |  E  ||
   flag      DNE  ||  E  |  C  |  I  | DNE |  NC |  E  ||
              NC  ||  E  |  NC |  NC | DNE |  NC |  E  ||
              NA  ||  E  |  E  |  E  |  E  |  E  |  E  ||
 -----------------||-----------------------------------||
                        </evaluation_chart>
      </xsd:appinfo>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="COMPLEMENT">
        <xsd:annotation>
          <xsd:documentation>The complement
            operator is defined in OVAL as a
            relative complement. The resulting
            unique 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, with the
            duplicates
            removed.</xsd:documentation>
        </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="INTERSECTION">
        <xsd:annotation>
          <xsd:documentation>The intersection of
```

```
                two sets in OVAL results in a unique
                set that contains everything that
                belongs to 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, with the duplicates
                removed.</xsd:documentation>
              </xsd:annotation>
            </xsd:enumeration>
```



```
            <xsd:enumeration value="UNION">
              <xsd:annotation>
                <xsd:documentation>The union of two sets
                  in OVAL results in a unique 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.</xsd:documentation>
              </xsd:annotation>
            </xsd:enumeration>
          </xsd:restriction>
        </xsd:simpleType>
      <!-- ========================================================= -->
      <!-- ==================  ENTITY TYPES  ======================= -->
      <!-- ========================================================= -->

        <xsd:attributeGroup name="EntityAttributeGroup">
          <xsd:annotation>
            <xsd:documentation>The EntityAttributeGroup
              is a collection of attributes that are
              common to all entities. This group defines
              these attributes and their default values.
              Individual entities may limit allowed
              values for these attributes, but all
              entities will support these
              attributes.</xsd:documentation>
            <xsd:appinfo>
              <sch:pattern
                id="oval-def_definition_entity_rules">
                <!-- These schematron rules are written to
                  look at object and state entities as well
                  as fields in states. -->
                <!-- var_ref and var_check rules -->
                <sch:rule
                  context="oval-def:objects/*/*[@var_ref]|
                  oval-def:objects/*/*/*[@var_ref]|
                  oval-def:states/*/*[@var_ref]|
                  oval-def:states/*/*/*[@var_ref]">
                  <sch:let name="var_ref"
                    value="@var_ref"/>
                  <sch:assert test=".=''"><sch:value-of
                      select="../@id"/> - a var_ref has
                    been supplied for the <sch:value-of
                      select="name()"/> entity so no
                    value should be
                    provided</sch:assert>
```



```
                  <sch:assert
                    test="( (not(@datatype)) and
                    ('string' = ancestor::oval-def:oval_definitions/
                    oval-def:variables/*[@id=$var_ref]/
                    @datatype) ) or
                    (@datatype = ancestor::oval-def:oval_definitions/
```

```
      oval-def:variables/*[@id=$var_ref]/
      @datatype)"
        ><sch:value-of select="$var_ref"/>
      - inconsistent datatype between the
      variable and an associated
      var_ref</sch:assert>
    </sch:rule>
    <sch:rule
      context="oval-def:objects/*/*[@var_ref]|
      oval-def:objects/*/*/*[@var_ref]">
      <sch:report test="not(@var_check)"
          ><sch:value-of select="../@id"/> -
        a var_ref has been supplied for the
          <sch:value-of select="name()"/>
        entity so a var_check should also be
        provided</sch:report>
    </sch:rule>
    <sch:rule
      context="oval-def:objects/*/*[@var_check]|
      oval-def:objects/*/*/*[@var_check]">
      <sch:assert test="@var_ref"
          ><sch:value-of select="../@id"/> -
        a var_check has been supplied for
        the <sch:value-of select="name()"/>
        entity so a var_ref must also be
        provided</sch:assert>
    </sch:rule>
    <sch:rule
      context="oval-def:states/*/*[@var_ref]|
      oval-def:states/*/*/*[@var_ref]">
      <sch:report test="not(@var_check)"
          ><sch:value-of select="../@id"/> -
        a var_ref has been supplied for the
          <sch:value-of select="name()"/>
        entity so a var_check should also be
        provided</sch:report>
    </sch:rule>
    <sch:rule
      context="oval-def:states/*/*[@var_check]|
      oval-def:states/*/*/*[@var_check]">
      <sch:assert test="@var_ref"
          ><sch:value-of select="../@id"/> -
```



```
        a var_check has been supplied for
        the <sch:value-of select="name()"/>
        entity so a var_ref must also be
        provided</sch:assert>
    </sch:rule>
    <!-- datatype and operation rules -->
    <sch:rule
      context="oval-def:objects/*/*[not(@datatype)]|
      oval-def:objects/*/*/*[not(@datatype)]|
      oval-def:states/*/*[not(@datatype)]|
      oval-def:states/*/*/*[not(@datatype)]">
      <sch:assert
        test="not(@operation) or
        @operation='equals' or
        @operation='not equal' or
        @operation='case insensitive equals' or
        @operation='case insensitive not equal' or
        @operation='pattern match'"
          ><sch:value-of select="../@id"/> -
        The use of '<sch:value-of
          select="@operation"/>' for the
        operation attribute of the
          <sch:value-of select="name()"/>
        entity is not valid given the lack
        of a declared datatype (hence a
        default datatype of
        string).</sch:assert>
    </sch:rule>
    <sch:rule
      context="oval-def:objects/*/*[@datatype='binary']|
      oval-def:objects/*/*/*[@datatype='binary']|
```

```
      oval-def:states/*/*[@datatype='binary']|
      oval-def:states/*/*/*[@datatype='binary']">
      <sch:assert
        test="not(@operation) or
        @operation='equals' or
        @operation='not equal'"
          ><sch:value-of select="../@id"/> -
        The use of '<sch:value-of
          select="@operation"/>' for the
        operation attribute of the
          <sch:value-of select="name()"/>
        entity is not valid given a datatype
        of binary.</sch:assert>
      <!--<sch:assert test=
        "matches(., '^[0-9a-fA-F]*$')">
        <sch:value-of select="../@id"/>
        - A value of '<sch:value-of select="."/>'
```



```
        for the <sch:value-of select="name()"/>
        entity is not valid given a datatype of binary.
        </sch:assert>-->
    </sch:rule>
    <sch:rule
      context="oval-def:objects/*/*[@datatype='boolean']|
      oval-def:objects/*/*/*[@datatype='boolean']|
      oval-def:states/*/*[@datatype='boolean']|
      oval-def:states/*/*/*[@datatype='boolean']">
      <sch:assert
        test="not(@operation) or
        @operation='equals' or
        @operation='not equal'"
          ><sch:value-of select="../@id"/> -
        The use of '<sch:value-of
          select="@operation"/>' for the
        operation attribute of the
          <sch:value-of select="name()"/>
        entity is not valid given a datatype
        of boolean.</sch:assert>
      <!--<sch:assert test="matches(., '^true$|
        ^false$|^1$|^0$')">
        <sch:value-of select="../@id"/>
        - A value of '<sch:value-of select="."/>'
        for the
        <sch:value-of select="name()"/>
        entity is not valid given a datatype of boolean.
        </sch:assert>-->
    </sch:rule>
    <sch:rule
      context="oval-def:objects/*/*
      [@datatype='evr_string']|
      oval-def:objects/*/*/*[@datatype='evr_string']|
      oval-def:states/*/*[@datatype='evr_string']|
      oval-def:states/*/*/*[@datatype='evr_string']">
      <sch:assert
        test="not(@operation) or
        @operation='equals' or
        @operation='not equal' or
        @operation='greater than' or
        @operation='greater than or
        equal' or @operation='less than' or
        @operation='less than or equal'"
          ><sch:value-of select="../@id"/> -
        The use of '<sch:value-of
          select="@operation"/>' for the
        operation attribute of the
          <sch:value-of select="name()"/>
```

```
      entity is not valid given a datatype
      of evr_string.</sch:assert>
    <!--<sch:assert test=
      "matches(., '^[^:\-]*:[^:\-]*-[^:\-]*$')">
      <sch:value-of select="../@id"/>
      - A value of '<sch:value-of select="."/>'
      for the <sch:value-of select="name()"/>
      entity is not valid given a datatype
      of evr_string.</sch:assert>-->
  </sch:rule>
  <sch:rule
    context="oval-def:objects/*/*
    [@datatype='debian_evr_string']|
    oval-def:objects/*/*/*
    [@datatype='debian_evr_string']|
    oval-def:states/*/*
    [@datatype='debian_evr_string']|
    oval-def:states/*/*/*
    [@datatype='debian_evr_string']">
    <sch:assert
      test="not(@operation) or
      @operation='equals' or
      @operation='not equal' or
      @operation='greater than' or
      @operation='greater than or
      equal' or
      @operation='less than' or
      @operation='less than or equal'"
        ><sch:value-of select="../@id"/> -
      The use of '<sch:value-of
        select="@operation"/>' for the
      operation attribute of the
        <sch:value-of select="name()"/>
      entity is not valid given a datatype
      of debian_evr_string.</sch:assert>
  </sch:rule>
  <sch:rule
    context="oval-def:objects/*/*
    [@datatype='fileset_revision']|
    oval-def:objects/*/*/*
    [@datatype='fileset_revision']|
    oval-def:states/*/*
    [@datatype='fileset_revision']|
    oval-def:states/*/*/*
    [@datatype='fileset_revision']">
    <sch:assert
      test="not(@operation) or
      @operation='equals' or
```



```
      @operation='not equal' or
      @operation='greater than' or
      @operation='greater than or
      equal' or @operation='less than' or
      @operation='less than or equal'"
        ><sch:value-of select="../@id"/> -
      The use of '<sch:value-of
        select="@operation"/>' for the
      operation attribute of the
        <sch:value-of select="name()"/>
      entity is not valid given a datatype
      of fileset_revision.</sch:assert>
  </sch:rule>
  <sch:rule
    context="oval-def:objects/*/*
    [@datatype='float']|
    oval-def:objects/*/*/*
    [@datatype='float']|
    oval-def:states/*/*
    [@datatype='float']|
    oval-def:states/*/*/*
    [@datatype='float']">
    <sch:assert
      test="not(@operation) or
      @operation='equals' or
```

```
                @operation='not equal' or
                @operation='greater than' or
                @operation='greater than or equal' or
                @operation='less than' or
                @operation='less than or equal'"
                  ><sch:value-of select="../@id"/> -
                The use of '<sch:value-of
                  select="@operation"/>' for the
                operation attribute of the
                  <sch:value-of select="name()"/>
                entity is not valid given a datatype
                of float.</sch:assert>
              <!--<sch:assert test="
                matches(., '^[+\-]?[0-9]+([\.][0-9]+)?
                ([eE][+\-]?[0-9]+)?$|^NaN$|^INF$|^\-INF$')">
                <sch:value-of select="../@id"/>
                - A value of '<sch:value-of select="."/>'
                for the <sch:value-of select="name()"/>
                entity is not valid given a datatype of
                float.</sch:assert>-->
            </sch:rule>
            <sch:rule
              context="oval-def:objects/*/*
```



```
              [@datatype='ios_version']|
              oval-def:objects/*/*/*
              [@datatype='ios_version']|
              oval-def:states/*/*
              [@datatype='ios_version']|
              oval-def:states/*/*/*
              [@datatype='ios_version']">
              <sch:assert
                test="not(@operation) or
                @operation='equals' or
                @operation='not equal' or
                @operation='greater than' or
                @operation='greater than or
                equal' or @operation='less than' or
                @operation='less than or equal'"
                  ><sch:value-of select="../@id"/> -
                The use of '<sch:value-of
                  select="@operation"/>' for the
                operation attribute of the
                  <sch:value-of select="name()"/>
                entity is not valid given a datatype
                of ios_version.</sch:assert>
            </sch:rule>
            <sch:rule
              context="oval-def:objects/*/*[@datatype='int']|
              oval-def:objects/*/*/*[@datatype='int']|
              oval-def:states/*/*[@datatype='int']|
              oval-def:states/*/*/*[@datatype='int']">
              <sch:assert
                test="not(@operation) or
                @operation='equals' or
                @operation='not equal' or
                @operation='greater than' or
                @operation='greater than or equal' or
                @operation='less than' or
                @operation='less than or equal' or
                @operation='bitwise and' or
                @operation='bitwise or'"
                  ><sch:value-of select="../@id"/> -
                The use of '<sch:value-of
                  select="@operation"/>' for the
                operation attribute of the
                  <sch:value-of select="name()"/>
                entity is not valid given a datatype
                of int.</sch:assert>
              <!--<sch:assert test="
                matches(., '^[+\-]?[0-9]+$')">
                <sch:value-of select="../@id"/>
```

```
                - A value of '<sch:value-of select="."/>' for
                the <sch:value-of select="name()"/>
                entity is not valid given a datatype
                of int.</sch:assert>-->
              </sch:rule>
              <sch:rule
                context="oval-def:objects/*/*
                [@datatype='ipv4_address']|
                oval-def:objects/*/*/*[@datatype='ipv4_address']|
                oval-def:states/*/*[@datatype='ipv4_address']|
                oval-def:states/*/*/*[@datatype='ipv4_address']">
                <sch:assert
                  test="not(@operation) or
                  @operation='equals' or
                  @operation='not equal' or
                  @operation='greater than' or
                  @operation='greater than or
                  equal' or @operation='less than' or
                  @operation='less than or equal' or
                  @operation='subset of' or
                  @operation='superset of'"
                    ><sch:value-of select="../@id"/> -
                  The use of '<sch:value-of
                    select="@operation"/>' for the
                  operation attribute of the
                    <sch:value-of select="name()"/>
                  entity is not valid given a datatype
                  of ipv4_address.</sch:assert>
                <!-- TODO <sch:assert test=
                  "matches(we_need_regex_for_ipv4)">
                  <sch:value-of select="../@id"/>
                  - A value of '<sch:value-of select="."/>'
                  for the <sch:value-of select="name()"/>
                  entity is not valid given a datatype of
                  ipv4_address.</sch:assert>-->
              </sch:rule>
              <sch:rule
                context="oval-def:objects/*/*
                [@datatype='ipv6_address']|
                oval-def:objects/*/*/*
                [@datatype='ipv6_address']|
                oval-def:states/*/*
                [@datatype='ipv6_address']|
                oval-def:states/*/*/*
                [@datatype='ipv6_address']">
                <sch:assert
                  test="not(@operation) or
                  @operation='equals' or
```



```
                  @operation='not equal' or
                  @operation='greater than' or
                  @operation='greater than or
                  equal' or @operation='less than' or
                  @operation='less than or equal' or
                  @operation='subset of' or
                  @operation='superset of'"
                    ><sch:value-of select="../@id"/> -
                  The use of '<sch:value-of
                    select="@operation"/>' for the
                  operation attribute of the
                    <sch:value-of select="name()"/>
                  entity is not valid given a datatype
                  of ipv6_address.</sch:assert>
                <!-- TODO <sch:assert test=
                  "matches(we_need_regex_for_ipv6)">
                  <sch:value-of select="../@id"/>
                  - A value of '<sch:value-of select="."/>'
                  for the <sch:value-of select="name()"/>
```

```
            entity is not valid given a datatype of
            ipv6_address.</sch:assert>-->
        </sch:rule>
        <sch:rule
          context="oval-def:objects/*/*
          [@datatype='string']|
          oval-def:objects/*/*/*
          [@datatype='string']|
          oval-def:states/*/*[@datatype='string']|
          oval-def:states/*/*/*[@datatype='string']">
          <sch:assert
            test="not(@operation) or
            @operation='equals' or
            @operation='not equal' or
            @operation='case insensitive equals' or
            @operation='case insensitive not equal' or
            @operation='pattern match'"
              ><sch:value-of select="../@id"/> -
            The use of '<sch:value-of
              select="@operation"/>' for the
            operation attribute of the
              <sch:value-of select="name()"/>
            entity is not valid given a datatype
            of string.</sch:assert>
        </sch:rule>
        <sch:rule
          context="oval-def:objects/*/*
          [@datatype='version']|oval-def:objects
          /*/*/*[@datatype='version']|oval-def:states/*/*
```



```
          [@datatype='version']|oval-def:states/*/*/*
          [@datatype='version']">
          <sch:assert
            test="not(@operation) or
            @operation='equals' or
            @operation='not equal' or
            @operation='greater than' or
            @operation='greater than or equal' or
            @operation='less than' or
            @operation='less than or equal'"
              ><sch:value-of select="../@id"/> -
            The use of '<sch:value-of
              select="@operation"/>' for the
            operation attribute of the
              <sch:value-of select="name()"/>
            entity is not valid given a datatype
            of version.</sch:assert>
        </sch:rule>
        <sch:rule
          context="oval-def:objects/*/*
          [@datatype='record']|oval-def:states/*/*
          [@datatype='record']">
          <sch:assert
            test="not(@operation) or @operation='equals'"
              ><sch:value-of select="../@id"/> -
            The use of '<sch:value-of
              select="@operation"/>' for the
            operation attribute of the
              <sch:value-of select="name()"/>
            entity is not valid given a datatype
            of record.</sch:assert>
        </sch:rule>
      </sch:pattern>
      <sch:pattern
        id="oval-def_no_var_ref_with_records">
        <sch:rule
          context="oval-def:objects/*/*
          [@datatype='record']|oval-def:states/*/*
          [@datatype='record']">
          <sch:assert test="not(@var_ref)"
              ><sch:value-of select="../@id"/> -
            The use of var_ref is prohibited
            when the datatype is
            'record'.</sch:assert>
```

```
            </sch:rule>
          </sch:pattern>
          <sch:pattern
            id="oval-def_definition_entity_type_check_rules">
```



```
            <sch:rule
              context="oval-def:objects/*/*
              [not((@xsi:nil='1' or @xsi:nil='true')) and
              not(@var_ref) and @datatype='int']|
              oval-def:objects/*/*/*[not(@var_ref) and
              @datatype='int']|oval-def:states/*/*
              [not((@xsi:nil='1' or @xsi:nil='true')) and
              not(@var_ref) and @datatype='int']|
              oval-def:states/*/*/*[not(@var_ref) and
              @datatype='int']">
              <sch:assert
                test="(not(contains(.,'.'))) and
                (number(.) = floor(.))"
                  ><sch:value-of select="../@id"/> -
                The datatype for the <sch:value-of
                  select="name()"/> entity is 'int'
                but the value is not an
                integer.</sch:assert>
              <!--  Must test for decimal point because
                number(x.0) = floor(x.0) is true -->
            </sch:rule>
          </sch:pattern>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:attribute name="datatype"
        type="oval:DatatypeEnumeration"
        use="optional" default="string">
        <xsd:annotation>
          <xsd:documentation>The optional datatype
            attribute specifies how the given
            operation should be applied to the data.
            Since we are dealing with XML everything
            is technically a string, but often the
            value is meant to represent some other
            datatype and this affects the way an
            operation is performed. For example,
            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 defines 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 (note
            that the default datatype is 'string').
            In the previous example, if the datatype
```



```
            is set to int, then '123' and '98'
            should be cast as integers. Another
            example is applying the 'equals'
            operation to '1.0.0.0' and '1.0'. With
            datatype 'string' they are not equal,
            with datatype 'version' they are. Note
            that there are certain cases where a
            cast from one datatype to another is not
            possible. If a cast cannot be made,
            (trying to cast 'abc' to an integer)
            then an error should be reported. For
            example, if the datatype is set to
            'integer' and the value is the empty
```

```
            string. There is no way to cast the
            empty string (or NULL) to an integer,
            and in cases like this an error should
            be reported.</xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
      <xsd:attribute name="operation"
        type="oval:OperationEnumeration"
        use="optional" default="equals">
        <xsd:annotation>
          <xsd:documentation>The optional operation
            attribute determines how the individual
            entities should be evaluated (the
            default operation is
            'equals').</xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
      <xsd:attribute name="mask" type="xsd:boolean"
        use="optional" default="false">
        <xsd:annotation>
          <xsd:documentation>The optional mask
            attribute is used to identify values
            that have been hidden for sensitivity
            concerns. This is used by the Result
            document which uses the System
            Characteristics schema to format the
            information found on a specific system.
            When the mask attribute is set to 'true'
            on an OVAL Entity or an OVAL Field, the
            corresponding collected value of that
            OVAL Entity or OVAL Field MUST NOT be
            present in the "results" section of the
            OVAL Results document; the
            "oval_definitions" section must not be
            altered and must be an exact copy of the
```



```
            definitions evaluated. Values MUST NOT
            be masked in OVAL System Characteristics
            documents that are not contained within
            an OVAL Results document. It is possible
            for masking conflicts to occur where one
            entity has mask set to true and another
            entity has mask set to false. A conflict
            will occur when the mask attribute is
            set differently on an OVAL Object and
            matching OVAL State or when more than
            one OVAL Objects identify the same OVAL
            Item(s). When such a conflict occurs the
            result is always to mask the
            entity.</xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
      <xsd:attribute name="var_ref"
        type="oval:VariableIDPattern" use="optional">
        <xsd:annotation>
          <xsd:documentation>The optional var_ref
            attribute refers the value of the
            element to a variable element. When
            supplied, the value(s) associated with
            the OVAL Variable should be used as the
            value(s) of the element. If there is an
            error computing the value of the
            variable, then that error should be
            passed up to the element 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 element
            is part of an object declaration, then
            the object element referencing it is
            considered to not exist. If the element
            is part of a state declaration, then the
            state element referencing it will
```

```
            evaluate to error.</xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="var_check"
          type="oval:CheckEnumeration" use="optional">
          <xsd:annotation>
            <xsd:documentation>The optional var_check
              attribute specifies how data collection
              or state evaluation should proceed when
              an element uses a var_ref attribute, and
```



```
              the associated variable defines more
              than one value. For example, if an
              object entity 'filename' with an
              operation of '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. As another example, if
              a state entity 'size' with an operation
              of 'less than' references a variable
              that has five different integer values,
              and the var_check attribute has a value
              of 'all', then the 'size' state entity
              evaluates to true only if the
              corresponding 'size' item entity is less
              than each of the five integers defined
              by the variable. If a variable does not
              have any value value when referenced by
              an OVAL Object the object should be
              considered to not exist. If a variable
              does not have any value when referenced
              by an OVAL State an error should be
              reported during OVAL analysis. When an
              OVAL State uses a var_ref, if both the
              state entity and a corresponding item
              entity are collections of values, the
              var_check is applied to each value of
              the item entity individually, and all
              must evaluate to true for the state
              entity to evaluate to true. In this
              condition, there is no value of
              var_check which enables an element-wise
              comparison, and so there is no way to
              determine whether the two entities are
              truly 'equal' in that sense. If var_ref
              is present but var_check is not, the
              element should be processed as if
              var_check has the value
              "all".</xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
      </xsd:attributeGroup>

      <xsd:complexType name="EntitySimpleBaseType"
        abstract="true">
        <xsd:annotation>
```



```
          <xsd:documentation>The EntitySimpleBaseType
            complex type is an abstract type that
            defines the default attributes associated
            with every simple 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.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:extension base="xsd:anySimpleType">
          <xsd:attributeGroup
            ref="oval-def:EntityAttributeGroup"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>

    <xsd:complexType name="EntityComplexBaseType"
      abstract="true">
      <xsd:annotation>
        <xsd:documentation>The EntityComplexBaseType
          complex type is an abstract type that
          defines the default attributes associated
          with every complex 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.</xsd:documentation>
      </xsd:annotation>
      <xsd:attributeGroup
        ref="oval-def:EntityAttributeGroup"/>
    </xsd:complexType>

    <xsd:complexType
      name="EntityObjectIPAddressType">
      <xsd:annotation>
        <xsd:documentation>The
          EntityObjectIPAddressType 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
```



```
          EntitySimpleBaseType. This specific type
          describes any IPv4/IPv6 address or address
          prefix.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:restriction
          base="oval-def:EntitySimpleBaseType">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
          </xsd:simpleType>
          <xsd:attribute name="datatype"
            use="required">
            <xsd:simpleType>
              <xsd:restriction
                base="oval:SimpleDatatypeEnumeration">
                <xsd:enumeration
                  value="ipv4_address"/>
                <xsd:enumeration
                  value="ipv6_address"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType
      name="EntityObjectIPAddressStringType">
      <xsd:annotation>
        <xsd:documentation>The
          EntityObjectIPAddressStringType 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
            EntitySimpleBaseType. This specific type
            describes any IPv4/IPv6 address, address
            prefix, or its string
            representation.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntitySimpleBaseType">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string"/>
            </xsd:simpleType>
            <xsd:attribute name="datatype"
              use="optional" default="string">
              <xsd:simpleType>
```



```
                <xsd:restriction
                  base="oval:SimpleDatatypeEnumeration">
                  <xsd:enumeration
                    value="ipv4_address"/>
                  <xsd:enumeration
                    value="ipv6_address"/>
                  <xsd:enumeration value="string"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType
        name="EntityObjectAnySimpleType">
        <xsd:annotation>
          <xsd:documentation>The
            EntityObjectAnySimpleType 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
            EntitySimpleBaseType. This specific type
            describes any simple
            data.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntitySimpleBaseType">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string"/>
            </xsd:simpleType>
            <xsd:attribute name="datatype"
              type="oval:SimpleDatatypeEnumeration"
              use="optional" default="string"/>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType name="EntityObjectBinaryType">
        <xsd:annotation>
          <xsd:documentation>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
            EntitySimpleBaseType. This specific type
            describes simple binary data. The empty
            string is also allowed when using a
```



```
            variable reference with an
```

```
          element.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntitySimpleBaseType">
            <xsd:simpleType>
              <xsd:union
                memberTypes="xsd:hexBinary oval:EmptyStringType"
              />
            </xsd:simpleType>
            <xsd:attribute name="datatype"
              type="oval:SimpleDatatypeEnumeration"
              use="required" fixed="binary"/>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType name="EntityObjectBoolType">
        <xsd:annotation>
          <xsd:documentation>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
            EntitySimpleBaseType. This specific type
            describes simple boolean data. The empty
            string is also allowed when using a
            variable reference with an
            element.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntitySimpleBaseType">
            <xsd:simpleType>
              <xsd:union
                memberTypes="xsd:boolean oval:EmptyStringType"
              />
            </xsd:simpleType>
            <xsd:attribute name="datatype"
              type="oval:SimpleDatatypeEnumeration"
              use="required" fixed="boolean"/>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType name="EntityObjectFloatType">
        <xsd:annotation>
          <xsd:documentation>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
            EntitySimpleBaseType. This specific type
            describes simple float data. The empty
            string is also allowed when using a
            variable reference with an
            element.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntitySimpleBaseType">
            <xsd:simpleType>
              <xsd:union
                memberTypes="xsd:float oval:EmptyStringType"
              />
            </xsd:simpleType>
            <xsd:attribute name="datatype"
              type="oval:SimpleDatatypeEnumeration"
              use="required" fixed="float"/>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType name="EntityObjectIntType">
        <xsd:annotation>
          <xsd:documentation>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
            EntitySimpleBaseType. This specific type
            describes simple integer data. The empty
            string is also allowed when using a
            variable reference with an
            element.</xsd:documentation>
          </xsd:annotation>
          <xsd:simpleContent>
            <xsd:restriction
              base="oval-def:EntitySimpleBaseType">
              <xsd:simpleType>
                <xsd:union
                  memberTypes="xsd:integer oval:EmptyStringType"
                />
              </xsd:simpleType>
              <xsd:attribute name="datatype"
                type="oval:SimpleDatatypeEnumeration"
                use="required" fixed="int"/>
            </xsd:restriction>
```



```
          </xsd:simpleContent>
        </xsd:complexType>
        <xsd:complexType name="EntityObjectStringType">
          <xsd:annotation>
            <xsd:documentation>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
              EntitySimpleBaseType. This specific type
              describes simple string
              data.</xsd:documentation>
          </xsd:annotation>
          <xsd:simpleContent>
            <xsd:restriction
              base="oval-def:EntitySimpleBaseType">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string"/>
              </xsd:simpleType>
              <xsd:attribute name="datatype"
                type="oval:SimpleDatatypeEnumeration"
                use="optional" fixed="string"/>
            </xsd:restriction>
          </xsd:simpleContent>
        </xsd:complexType>
        <xsd:complexType name="EntityObjectVersionType">
          <xsd:annotation>
            <xsd:documentation>The
              EntityObjectVersionType 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
              EntityStateSimpleBaseType. This specific
              type describes simple version
              data.</xsd:documentation>
          </xsd:annotation>
          <xsd:simpleContent>
            <xsd:restriction
              base="oval-def:EntitySimpleBaseType">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string"/>
              </xsd:simpleType>
              <xsd:attribute name="datatype"
                type="oval:SimpleDatatypeEnumeration"
                use="required" fixed="version"/>
            </xsd:restriction>
          </xsd:simpleContent>
```

```
    </xsd:complexType>
    <xsd:complexType name="EntityObjectRecordType">
      <xsd:annotation>
        <xsd:documentation>The
          EntityObjectRecordType defines an entity
          that consists of a number of uniquely
          named fields. This structure is used for
          representing a record from a database
          query and other similar structures where
          multiple related fields must be
          represented at once. Note that for all
          entities of this type, the only allowed
          datatype is 'record' and the only allowed
          operation is 'equals'. During analysis of
          a system characteristics item, each field
          is analyzed and then the overall result
          for elements of this type is computed by
          logically anding the results for each
          field and then applying the entity_check
          attribute.</xsd:documentation>
        <xsd:documentation>Note the datatype
          attribute must be set to
          'record'.</xsd:documentation>
        <!--
                  NOTE: The restriction that the only
                  allowed datatype is 'record' is enforced
                  by scheamtron rules placed on each
                  entity that uses this type.  This is
                  due to the fact that this type is
                  developed as an extension of the
                  oval-def:EntityComplexBaseType.
                  This base type declares a datatype
                  attribute. to restrict the datatype
                  attribute to only alloy 'record'
                  would need a restriction. We
                  cannot do both and xsd:extension
                  and an xsd:restriction at the same
                  time.
                -->
        <xsd:documentation>Note the operation
          attribute must be set to
          'equals'.</xsd:documentation>
        <xsd:documentation>Note the var_ref
          attribute is not permitted and the
          var_check attribute does not
          apply.</xsd:documentation>
        <xsd:documentation>Note that when the mask
          attribute is set to 'true', all child
```



```
          field elements must be masked regardless
          of the child field's mask attribute
          value.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
        <xsd:extension
          base="oval-def:EntityComplexBaseType">
          <xsd:sequence>
            <xsd:element name="field"
              type="oval-def:EntityObjectFieldType"
              minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="EntityObjectFieldType">
      <xsd:annotation>
        <xsd:documentation>The EntityObjectFieldType
          defines an element with simple content
          that represents a named field in a record
```

```
            that may contain any number of named
            fields. The EntityObjectFieldType is much
            like all other entities with one
            significant difference, the
            EntityObjectFieldType has a name
            attribute</xsd:documentation>
          <xsd:documentation>The required name
            attribute specifies a unique name for the
            field. Field names are lowercase and must
            be unique within a given parent record
            element. When analyzing system
            characteristics an error should be
            reported for the result of a field that is
            present in the OVAL State, but not found
            in the system characteristics
            Item.</xsd:documentation>
          <xsd:documentation>The optional entity_check
            attribute specifies how to handle multiple
            record fields with the same name in the
            OVAL Systems Characteristics file. For
            example, while collecting group
            information where one field is the
            represents the users that are members of
            the group. It is very likely that there
            will be multiple fields with a name of
            'user' associated with the group. If the
            OVAL State defines the value of the field
            with name equal 'user' to equal 'Fred',
```



```
            then the entity_check attribute determines
            if all values for field entities must be
            equal to 'Fred', or at least one value
            must be equal to 'Fred',
            etc.</xsd:documentation>
          <xsd:documentation>Note that when the mask
            attribute is set to 'true' on a field's
            parent element the field must be masked
            regardless of the field's mask attribute
            value.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:extension base="xsd:anySimpleType">
            <xsd:attribute name="name" use="required">
              <xsd:annotation>
                <xsd:documentation>A string restricted
                  to disallow upper case
                  characters.</xsd:documentation>
              </xsd:annotation>
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:pattern value="[^A-Z]+"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
            <xsd:attributeGroup
              ref="oval-def:EntityAttributeGroup"/>
            <xsd:attribute name="entity_check"
              type="oval:CheckEnumeration"
              use="optional" default="all"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>

      <xsd:complexType
        name="EntityStateSimpleBaseType"
        abstract="true">
        <xsd:annotation>
          <xsd:documentation>The
            EntityStateSimpleBaseType complex type is
            an abstract type that extends the
            EntitySimpleBaseType and is used by some
            entities within an OVAL
            State.</xsd:documentation>
          <xsd:documentation>The optional
```

```
          check_existence attribute specifies how to
          interpret the status of corresponding item
          entities when performing an item-state
```



```
          comparison. The default value for this
          attribute is 'at_least_one_exists'
          indicating that by default an item
          comparison may evaluate to true only if at
          least one corresponding item entity has a
          status of 'exists'. For example, if a
          value of 'none_exist' is given, then the
          comparison can evaluate to true only if
          there are one or more corresponding item
          entities, each with a status of 'does not
          exist'.</xsd:documentation>
        <xsd:documentation>The optional entity_check
          attribute specifies how to handle multiple
          item entities with the same name in the
          OVAL Systems Characteristics file. For
          example, suppose we are dealing with a
          Group Test and an entity in the state is
          related to the user. It is very likely
          that when the information about the group
          is collected off of the system (and
          represented in the OVAL System
          Characteristics file) that there will be
          multiple users associated with the group
          (i.e. multiple 'user' item entities
          associated with the same 'user' state
          entity). If the OVAL State defines the
          value of the user entity to equal 'Fred',
          then the entity_check attribute determines
          if all values for 'user' item entities
          must be equal to 'Fred', or at least one
          value must be equal to 'Fred', etc. Note
          that with the exception of the
          'none_satisfy' check value, the
          entity_check attribute can only affect the
          result of the test if the corresponding
          OVAL Item allows more than one occurrence
          of the entity (e.g. 'maxOccurs' is some
          value greater than
          one).</xsd:documentation>
        <xsd:documentation>The entity_check and
          var_check attributes are considered
          together when evaluating a single state
          entity. When a variable identifies more
          than one value and multiple item entities
          with the same name exist, for a single
          state entity, a many-to-many comparison
          must be conducted. In this situation,
          there are many values for the state entity
```



```
          that must be compared to many item
          entities. Each item entity is compared to
          the state entity. For each item entity, an
          interim result is calculated by using the
          var_check attribute to combine the result
          of comparing each variable value with a
          single system value. Then these interim
          results are combined for each system value
          using the entity_check
          attribute.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:extension
          base="oval-def:EntitySimpleBaseType">
```

```
            <xsd:attribute name="entity_check"
              type="oval:CheckEnumeration"
              use="optional" default="all"/>
            <xsd:attribute name="check_existence"
              type="oval:ExistenceEnumeration"
              use="optional"
              default="at_least_one_exists"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType
        name="EntityStateComplexBaseType"
        abstract="true">
        <xsd:annotation>
          <xsd:documentation>The
            EntityStateComplexBaseType complex type is
            an abstract type that extends the
            EntityComplexBaseType and is used by some
            entities within an OVAL
            State.</xsd:documentation>
          <xsd:documentation>The optional
            check_existence attribute specifies how to
            interpret the status of corresponding item
            entities when performing an item-state
            comparison. The default value for this
            attribute is 'at_least_one_exists'
            indicating that by default an item
            comparison may evaluate to true only if at
            least one corresponding item entity has a
            status of 'exists'. For example, if a
            value of 'none_exist' is given, then the
            comparison can evaluate to true only if
            there are one or more corresponding item
            entities, each with a status of 'does not
```



```
            exist'.</xsd:documentation>
          <xsd:documentation>The optional entity_check
            attribute specifies how to handle multiple
            item entities with the same name in the
            OVAL Systems Characteristics file. For
            example, suppose we are dealing with a
            Group Test and an entity in the state is
            related to the user. It is very likely
            that when the information about the group
            is collected off of the system (and
            represented in the OVAL System
            Characteristics file) that there will be
            multiple users associated with the group
            (i.e. multiple 'user' item entities
            associated with the same 'user' state
            entity). If the OVAL State defines the
            value of the user entity to equal 'Fred',
            then the entity_check attribute determines
            if all values for 'user' item entities
            must be equal to 'Fred', or at least one
            value must be equal to 'Fred', etc. Note
            that with the exception of the
            'none_satisfy' check value, the
            entity_check attribute can only affect the
            result of the test if the corresponding
            OVAL Item allows more than one occurrence
            of the entity (e.g. 'maxOccurs' is some
            value greater than
            one).</xsd:documentation>
          <xsd:documentation>The entity_check and
            var_check attributes are considered
            together when evaluating a single state
            entity. When a variable identifies more
            than one value and multiple item entities
            with the same name exist, for a single
            state entity, a many-to-many comparison
            must be conducted. In this situation,
            there are many values for the state entity
            that must be compared to many item
```

```
            entities. Each item entity is compared to
            the state entity. For each item entity, an
            interim result is calculated by using the
            var_check attribute to combine the result
            of comparing each variable value with a
            single system value. Then these interim
            results are combined for each system value
            using the entity_check
            attribute.</xsd:documentation>
```



```
        </xsd:annotation>
        <xsd:complexContent>
          <xsd:extension
            base="oval-def:EntityComplexBaseType">
            <xsd:attribute name="entity_check"
              type="oval:CheckEnumeration"
              use="optional" default="all"/>
            <xsd:attribute name="check_existence"
              type="oval:ExistenceEnumeration"
              use="optional"
              default="at_least_one_exists"/>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="EntityStateIPAddressType">
        <xsd:annotation>
          <xsd:documentation>The
            EntityStateIPAddressType type is extended
            by the entities of an individual OVAL
            State. This type provides uniformity to
            each object entity by including the
            attributes found in the
            EntityStateSimpleBaseType. This specific
            type describes any IPv4/IPv6 address or
            address prefix.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntityStateSimpleBaseType">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string"/>
            </xsd:simpleType>
            <xsd:attribute name="datatype"
              use="required">
              <xsd:simpleType>
                <xsd:restriction
                  base="oval:SimpleDatatypeEnumeration">
                  <xsd:enumeration
                    value="ipv4_address"/>
                  <xsd:enumeration
                    value="ipv6_address"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType
```



```
        name="EntityStateIPAddressStringType">
        <xsd:annotation>
          <xsd:documentation>The
            EntityStateIPAddressStringType type is
            extended by the entities of an individual
            OVAL State. This type provides uniformity
            to each object entity by including the
            attributes found in the
```

```
            EntityStateSimpleBaseType. This specific
            type describes any IPv4/IPv6 address,
            address prefix, or its string
            representation.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntityStateSimpleBaseType">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string"/>
            </xsd:simpleType>
            <xsd:attribute name="datatype"
              use="optional" default="string">
              <xsd:simpleType>
                <xsd:restriction
                  base="oval:SimpleDatatypeEnumeration">
                  <xsd:enumeration
                    value="ipv4_address"/>
                  <xsd:enumeration
                    value="ipv6_address"/>
                  <xsd:enumeration value="string"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType name="EntityStateAnySimpleType">
        <xsd:annotation>
          <xsd:documentation>The
            EntityStateAnySimpleType 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
            EntityStateSimpleBaseType. This specific
            type describes any simple
            data.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
```



```
          <xsd:restriction
            base="oval-def:EntityStateSimpleBaseType">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string"/>
            </xsd:simpleType>
            <xsd:attribute name="datatype"
              type="oval:SimpleDatatypeEnumeration"
              use="optional" default="string"/>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType name="EntityStateBinaryType">
        <xsd:annotation>
          <xsd:documentation>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
            EntityStateSimpleBaseType. This specific
            type describes simple binary data. The
            empty string is also allowed when using a
            variable reference with an
            element.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntityStateSimpleBaseType">
            <xsd:simpleType>
              <xsd:union
                memberTypes="xsd:hexBinary oval:EmptyStringType"
              />
            </xsd:simpleType>
            <xsd:attribute name="datatype"
```

```
            type="oval:SimpleDatatypeEnumeration"
            use="required" fixed="binary"/>
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="EntityStateBoolType">
      <xsd:annotation>
        <xsd:documentation>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
          EntityStateSimpleBaseType. This specific
          type describes simple boolean data. The
          empty string is also allowed when using a
```



```
          variable reference with an
          element.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:restriction
          base="oval-def:EntityStateSimpleBaseType">
          <xsd:simpleType>
            <xsd:union
              memberTypes="xsd:boolean oval:EmptyStringType"
            />
          </xsd:simpleType>
          <xsd:attribute name="datatype"
            type="oval:SimpleDatatypeEnumeration"
            use="required" fixed="boolean"/>
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="EntityStateFloatType">
      <xsd:annotation>
        <xsd:documentation>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
          EntityStateSimpleBaseType. This specific
          type describes simple float data. The
          empty string is also allowed when using a
          variable reference with an
          element.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:restriction
          base="oval-def:EntityStateSimpleBaseType">
          <xsd:simpleType>
            <xsd:union
              memberTypes="xsd:float oval:EmptyStringType"
            />
          </xsd:simpleType>
          <xsd:attribute name="datatype"
            type="oval:SimpleDatatypeEnumeration"
            use="required" fixed="float"/>
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="EntityStateIntType">
      <xsd:annotation>
        <xsd:documentation>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
          EntityStateSimpleBaseType. This specific
          type describes simple integer data. The
          empty string is also allowed when using a
          variable reference with an
          element.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:restriction
          base="oval-def:EntityStateSimpleBaseType">
          <xsd:simpleType>
            <xsd:union
              memberTypes="xsd:integer oval:EmptyStringType"
            />
          </xsd:simpleType>
          <xsd:attribute name="datatype"
            type="oval:SimpleDatatypeEnumeration"
            use="required" fixed="int"/>
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="EntityStateEVRStringType">
      <xsd:annotation>
        <xsd:documentation>The
          EntityStateEVRStringType 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
          EntityStateSimpleBaseType. This type
          represents the epoch, version, and release
          fields, for an RPM package, as a single
          version string. It has the form
          "EPOCH:VERSION-RELEASE". Note that a null
          epoch (or '(none)' as returned by rpm) is
          equivalent to '0' and would hence have the
          form 0:VERSION-RELEASE. Comparisons
          involving this datatype should follow the
          algorithm of librpm's rpmvercmp()
          function.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:restriction
          base="oval-def:EntityStateSimpleBaseType">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
```



```
            <!-- TODO: Should there be a pattern
              restriction here to enforce the
              pattern mentioned above? -->
          </xsd:simpleType>
          <xsd:attribute name="datatype"
            type="oval:SimpleDatatypeEnumeration"
            use="required" fixed="evr_string"/>
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType
      name="EntityStateDebianEVRStringType">
      <xsd:annotation>
        <xsd:documentation>The
          EntityStateDebianEVRStringType 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
          EntityStateSimpleBaseType. This type
          represents the epoch, upstream_version,
          and debian_revision fields, for a Debian
          package, as a single version string. It
          has the form
          "EPOCH:UPSTREAM_VERSION-DEBIAN_REVISION".
          Note that a null epoch (or '(none)' as
          returned by dpkg) is equivalent to '0' and
```

```
            would hence have the form
            0:UPSTREAM_VERSION-DEBIAN_REVISION.
            Comparisons involving this datatype should
            follow the algorithm outlined in Chapter 5
            of the "Debian Policy Manual"
            (https://www.debian.org/doc/debian-policy/
            ch-controlfields.html#s-f-Version).
            An implementation of this is the
            cmpversions() function in dpkg's
            enquiry.c.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntityStateSimpleBaseType">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string"/>
              <!-- TODO: Should there be a pattern
                restriction here to enforce the
                pattern mentioned above? -->
            </xsd:simpleType>
            <xsd:attribute name="datatype"
```



```
              type="oval:SimpleDatatypeEnumeration"
              use="required" fixed="debian_evr_string"
            />
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType name="EntityStateVersionType">
        <xsd:annotation>
          <xsd:documentation>The
            EntityStateVersionType 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 EntityStateSimpleBaseType.
            This specific type describes simple
            version data.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntityStateSimpleBaseType">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string"/>
            </xsd:simpleType>
            <xsd:attribute name="datatype"
              type="oval:SimpleDatatypeEnumeration"
              use="required" fixed="version"/>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
      <xsd:complexType
        name="EntityStateFileSetRevisionType">
        <xsd:annotation>
          <xsd:documentation>The
            EntityStateFileSetRevisionType 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
            EntityStateSimpleBaseType. This specific
            type represents the version string related
            to filesets in HP-UX.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:restriction
            base="oval-def:EntityStateSimpleBaseType">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string"/>
            </xsd:simpleType>
```

```
          <xsd:attribute name="datatype"
            type="oval:SimpleDatatypeEnumeration"
            use="required" fixed="fileset_revision"
          />
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType
      name="EntityStateIOSVersionType">
      <xsd:annotation>
        <xsd:documentation>The
          EntityStateIOSVersionType 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
          EntityStateSimpleBaseType. This specific
          type represents the version string related
          to CISCO IOS.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:restriction
          base="oval-def:EntityStateSimpleBaseType">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
          </xsd:simpleType>
          <xsd:attribute name="datatype"
            use="optional" default="string">
            <xsd:simpleType>
              <xsd:restriction
                base="oval:SimpleDatatypeEnumeration">
                <xsd:enumeration value="ios_version"/>
                <xsd:enumeration value="string">
                  <xsd:annotation>
                    <xsd:documentation>'string' is
                      included to allow for regular
                      expressions on IOS version
                      strings.</xsd:documentation>
                  </xsd:annotation>
                </xsd:enumeration>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="EntityStateStringType">
      <xsd:annotation>
```



```
        <xsd:documentation>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
          EntityStateSimpleBaseType. This specific
          type describes simple string
          data.</xsd:documentation>
      </xsd:annotation>
      <xsd:simpleContent>
        <xsd:restriction
          base="oval-def:EntityStateSimpleBaseType">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
          </xsd:simpleType>
          <xsd:attribute name="datatype"
            type="oval:SimpleDatatypeEnumeration"
            use="optional" fixed="string"/>
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
```

```
    <xsd:complexType name="EntityStateRecordType">
      <xsd:annotation>
        <xsd:documentation>The EntityStateRecordType
          defines an entity that consists of a
          number of uniquely named fields. This
          structure is used for representing a
          record from a database query and other
          similar structures where multiple related
          fields must be collected at once. Note
          that for all entities of this type, the
          only allowed datatype is 'record' and the
          only allowed operation is 'equals'. During
          analysis of a system characteristics item,
          each field is analyzed and then the
          overall result for elements of this type
          is computed by logically anding the
          results for each field and then applying
          the entity_check
          attribute.</xsd:documentation>
        <xsd:documentation>Note the datatype
          attribute must be set to
          'record'.</xsd:documentation>
        <!--
                  NOTE: The restriction that the only
                  allowed datatype is 'record' is
                  enforced by scheamtron rules placed
                  on each entity that uses this type.
```



```
                  This is due to the fact that this
                  type is developed as an extension
                  of the
                  oval-def:EntityStateComplexBaseType.
                  This base type declares a datatype
                  attribute. to restrict the
                  datatype attribute to only allow
                  'record' would need a restriction. We
                  cannot do both and xsd:extension and
                  an xsd:restriction at the same time.
              -->
        <xsd:documentation>Note the operation
          attribute must be set to
          'equals'.</xsd:documentation>
        <xsd:documentation>Note the var_ref
          attribute is not permitted and the
          var_check attribute does not
          apply.</xsd:documentation>
        <xsd:documentation>Note that when the mask
          attribute is set to 'true', all child
          field elements must be masked regardless
          of the child field's mask attribute
          value.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
        <xsd:extension
          base="oval-def:EntityStateComplexBaseType">
          <xsd:sequence>
            <xsd:element name="field"
              type="oval-def:EntityStateFieldType"
              minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="EntityStateFieldType">
      <xsd:annotation>
        <xsd:documentation>The EntityStateFieldType
          defines an element with simple content
          that represents a named field in a record
          that may contain any number of named
          fields. The EntityStateFieldType is much
          like all other entities with one
          significant difference, the
          EntityStateFieldType has a name
          attribute</xsd:documentation>
```

```
          <xsd:documentation>The required name
            attribute specifies a unique name for the
```



```
            field. Field names are lowercase and must
            be unique within a given parent record
            element. When analyzing system
            characteristics an error should be
            reported for the result of a field that is
            present in the OVAL State, but not found
            in the system characteristics
            Item.</xsd:documentation>
          <xsd:documentation>The optional entity_check
            attribute specifies how to handle multiple
            record fields with the same name in the
            OVAL Systems Characteristics file. For
            example, while collecting group
            information where one field is the
            represents the users that are members of
            the group. It is very likely that there
            will be multiple fields with a name of
            'user' associated with the group. If the
            OVAL State defines the value of the field
            with name equal 'user' to equal 'Fred',
            then the entity_check attribute determines
            if all values for field entities must be
            equal to 'Fred', or at least one value
            must be equal to 'Fred',
            etc.</xsd:documentation>
          <xsd:documentation>Note that when the mask
            attribute is set to 'true' on a field's
            parent element the field must be masked
            regardless of the field's mask attribute
            value.</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
          <xsd:extension base="xsd:anySimpleType">
            <xsd:attribute name="name" use="required">
              <xsd:annotation>
                <xsd:documentation>A string restricted
                  to disallow upper case
                  characters.</xsd:documentation>
              </xsd:annotation>
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:pattern value="[^A-Z]+"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
            <xsd:attributeGroup
              ref="oval-def:EntityAttributeGroup"/>
            <xsd:attribute name="entity_check"
```



```
              type="oval:CheckEnumeration"
              use="optional" default="all"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:schema>
```

83. Intellectual Property Considerations

DHS, on behalf of the United States, owns the registered OVAL trademarks, identifying the OVAL STANDARDS SUITE and any component

part, as that suite has been provided to the IETF Trust. A "(R)" will be used in conjunction with the first use of any OVAL trademark in any document or publication in recognition of DHS's trademark ownership.

84. Acknowledgements

The authors wish to thank DHS for sponsoring the OVAL effort over the years which has made this work possible. The authors also wish to thank the original authors of this document Jonathan Baker, Matthew Hansbury, and Daniel Haynes of the MITRE Corporation as well as the OVAL Community for its assistance in contributing and reviewing the original document. The authors would also like to acknowledge Dave Waltermire of NIST for his contribution to the development of the original document.

85. IANA Considerations

This memo includes no request to IANA.

86. Security Considerations

While OVAL is just a set of data models and does not directly introduce security concerns, it does provide a mechanism by which to represent endpoint posture assessment information. This information could be extremely valuable to an attacker allowing them to learn about very sensitive information including, but, not limited to: security policies, systems on the network, criticality of systems, software and hardware inventory, patch levels, user accounts and much more. To address this concern, all endpoint posture assessment information should be protected while in transit and at rest. Furthermore, it should only be shared with parties that are authorized to receive it.

Another possible security concern is due to the fact that content expressed as OVAL has the ability to impact how a security tool operates. For example, content may instruct a tool to collect certain information off a system or may be used to drive follow-up actions like remediation. As a result, it is important for security tools to ensure that they are obtaining OVAL content from a trusted source, that it has not been modified in transit, and that proper validation is performed in order to ensure it does not contain malicious data.

87. Change Log

87.1. -00 to -01

There are no textual changes associated with this revision. This revision simply reflects a resubmission of the document so that it remains in active status.

88. References

88.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[WIN-FILETIME] Microsoft Corporation, "File Times", 2015, <https://msdn.microsoft.com/en-us/library/ms724290(v=vs.85).aspx>.

88.2. Informative References

[OVAL-WEBSITE] The MITRE Corporation, "The Open Vulnerability and Assessment Language", 2015, <http://ovalproject.github.io/>.

Authors' Addresses

Michael Cokus
The MITRE Corporation
903 Enterprise Parkway, Suite 200
Hampton, VA 23666
USA

Email: msc@mitre.org

Daniel Haynes
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730
USA

Email: dhaynes@mitre.org

David Rothenberg
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730
USA

Email: drothenberg@mitre.org

Juan Gonzalez
Department of Homeland Security
245 Murray Lane
Washington, DC 20548
USA

Email: juan.gonzalez@dhs.gov