THE MITRE CORPORATION

The OVAL® Language Specification

Version 5.11

Jonathan Baker, Matthew Hansbury, Daniel Haynes

12/18/2014

Information security is a function that consumes significant organizational resources, and is growing increasingly difficult to manage. One of the biggest problems is the lack of standardization between the sources of security information, and the tools that consume that information, as well as between the various tools themselves. Often, the exchange of security information is time critical, but is hampered by the variety of incompatible formats in which it is represented. The Open Vulnerability and Assessment Language (OVAL®) is an international, information security, community standard to promote open and publicly available security content, and to standardize the transfer of this information across the entire spectrum of security tools and services. By standardizing the three main steps of the assessment process: representing configuration information of systems for testing; analyzing the system for the presence of the specified machine state; and reporting the results of the assessment, the OVAL Language provides a common and structured format that facilitates collaboration and information sharing among the information security community as well as interoperability among tools. This document defines the use cases, requirements, data model, and processing model for the OVAL Language.

Acknowledgements

The authors, Jonathan Baker, Matthew Hansbury, and Daniel Haynes of the MITRE Corporation wish to thank the OVAL Community for its assistance in contributing and reviewing this document. The authors would like to acknowledge Dave Waltermire of NIST for his contribution to the development of this document.

Trademark Information

Warnings

MITRE PROVIDES OVAL "AS IS" AND MAKES NO WARRANTY, EXPRESS OR IMPLIED, AS TO THE ACCURACY, CAPABILITY, EFFICIENCY, MERCHANTABILITY, OR FUNCTIONING OF OVAL. IN NO EVENT WILL MITRE BE LIABLE FOR ANY GENERAL, CONSEQUENTIAL, INDIRECT, INCIDENTAL, EXEMPLARY, OR SPECIAL DAMAGES, RELATED TO OVAL OR ANY DERIVATIVE THEREOF, WHETHER SUCH CLAIM IS BASED ON WARRANTY, CONTRACT, OR TORT, EVEN IF MITRE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.[1]

Feedback

The MITRE Corporation welcomes any feedback regarding the OVAL Language Specification. Please send any comments, questions, or suggestions to the public OVAL Developer's Forum at oval-developer-list@lists.mitre.org or directly to the OVAL Moderator at oval@mitre.org.[2]

[1] For detailed information see https://oval.mitre.org/about/termsofuse.html
[2] For more information about the OVAL Language, please visit https://oval.mitre.org/

Table of Contents

Acknowledgements 2
Trademark Information 2
Warnings 2
Feedback 2
1 Introduction 12
1.1 The OVAL Language 13
1.2 Document Conventions 13
1.3 Document Structure 14
2 Use Cases for the OVAL Language 15
2.1 Security Advisory Distribution 15
Use Case Scenario: Publishing an Advisory 15
2.2 Vulnerability Management 16
Use Case Scenario: Leveraging a Standardized Security Advisory 17
Use Case Scenario: Collaborating on the Development of a Vulnerability Check 17
Use Case Scenario: Sharing Vulnerability Assessment Results 17
2.3 Patch Management 17
Use Case Scenario: Leveraging a Standardized Patch Check 18
Use Case Scenario: Patching a Known Vulnerability 18
2.4 Configuration Management 18
Use Case Scenario: Configuration Guidance Distribution 19
Use Case Scenario: Authoritative Policy Reuse 19
Use Case Scenario: Compliance Reporting 20
2.5 System Inventory 20
Use Case Scenario: Operating System Upgrade 20
2.6 Malware Artifact Hunting 21
Use Case Scenario: Detecting Compromised Systems 21
Use Case Scenario: Sharing Checks for Threat Indicators 21
2.7 Network Access Control (NAC) 22
Use Case Scenario: Minimum Secure Configuration Baseline Enforcement 22
2.8 Auditing and Centralized Audit Validation 22

Use Case Scenario: Keeping Track of Change 22
2.9 Security Information Management Systems (SIMS) 23
Use Case Scenario: Data Aggregation 23
3 Requirements for the OVAL Language 23
3.1 Basic Requirements 23
3.1.1 Expressing Expected Configuration State 23
3.1.2 Representing Observed Configuration State 23
3.1.3 Expressing Assessment Results 23
3.1.4 Content Integrity and Authenticity 23
3.2 Detailed Requirements 24
3.2.1 General Content Requirements 24
3.2.2 OVAL Definition Requirements 24
3.2.3 OVAL System Characteristics Requirements 24
3.2.4 OVAL Results Requirements 25
4 Data Model for the OVAL Language 25
4.1 Data Model Conventions 27
4.1.1 UML Diagrams 27
4.1.2 Property Table Notation 27
4.1.3 Primitive Data Types 28
4.2 OVAL Common Model 28
4.2.1 GeneratorType 28
4.2.2 MessageType 29
4.2.3 CheckEnumeration 29
4.2.4 ClassEnumeration 29
4.2.5 SimpleDatatypeEnumeration 30
int 32
ipv4_address 32
4.2.6 ComplexDatatypeEnumeration 33
4.2.7 DatatypeEnumeration 34
4.2.8 ExistenceEnumeration 34
4.2.9 FamilyEnumeration 34

4.2.10 MessageLevelEnumeration 34
4.2.11 OperationEnumeration 35
4.2.12 OperatorEnumeration 36
4.2.13 Definition, Test, Object, State, and Variable Identifiers 36
4.2.14 ItemIDPattern 37
4.2.15 EmptyStringType 37
4.2.16 NonEmptyStringType 37
4.2.17 Any 37
4.2.18 Signature 38
4.3 OVAL Definitions Model 38
4.3.1 oval_definitions 38
4.3.2 DefinitionsType 39
4.3.3 DefinitionType 39
4.3.4 MetadataType 40
4.3.5 AffectedType 40
4.3.6 ReferenceType 41
4.3.7 NotesType 41
4.3.8 CriteriaType 41
4.3.9 CriterionType 42
4.3.10 ExtendDefinitionType 43
4.3.11 TestsType 43
4.3.12 TestType 43
4.3.13 ObjectRefType 45
4.3.14 StateRefType 45
4.3.15 ObjectsType 45
4.3.16 ObjectType 45
4.3.17 set 46
4.3.18 filter 47
4.3.19 StatesType 47
4.3.20 StateType 47
4.3.21 VariablesType 48

4.3.22 VariableType 48
4.3.23 external_variable 49
4.3.24 PossibleValueType 49
4.3.25 PossibleRestrictionType 50
4.3.26 RestrictionType 50
4.3.27 constant_variable 50
4.3.28 ValueType 50
4.3.29 local_variable 51
4.3.30 ComponentGroup 51
4.3.31 LiteralComponentType 52
4.3.32 ObjectComponentType 52
4.3.33 VariableComponentType 52
4.3.34 FunctionGroup 53
4.3.35 ArithmeticFunctionType 54
4.3.36 BeginFunctionType 54
4.3.37 ConcatFunctionType 55
4.3.38 CountFunctionType 55
4.3.39 EndFunctionType 55
4.3.40 EscapeRegexFunctionType 56
4.3.41 SplitFunctionType 56
4.3.42 SubstringFunctionType 57
4.3.43 TimeDifferenceFunctionType 57
4.3.44 UniqueFunctionType 58
4.3.45 RegexCaptureFunctionType 58
4.3.46 ArithmeticEnumeration 59
4.3.47 DateTimeFormatEnumeration 59
4.3.48 FilterActionEnumeration 60
4.3.49 SetOperatorEnumeration 60
4.3.50 EntityAttributeGroup 60
4.3.51 EntitySimpleBaseType 61
4.3.52 EntityComplexBaseType 61

4.3.53 EntityObjectIPAddressType 61
4.3.54 EntityObjectIPAddressStringType 62
4.3.55 EntityObjectAnySimpleType 62
4.3.56 EntityObjectBinaryType 62
4.3.57 EntityObjectBoolType 62
4.3.58 EntityObjectFloatType 63
4.3.59 EntityObjectIntType 63
4.3.60 EntityObjectStringType 63
4.3.61 EntityObjectRecordType 63
4.3.62 EntityObjectFieldType 64
4.3.63 EntityStateSimpleBaseType 64
4.3.64 EntityStateComplexBaseType 65
4.3.65 EntityStateIPAddressType 65
4.3.66 EntityStateIPAddressStringType 65
4.3.67 EntityStateAnySimpleType 65
4.3.68 EntityStateBinaryType 65
4.3.69 EntityStateBoolType 66
4.3.70 EntityStateFloatType 66
4.3.71 EntityStateIntType 66
4.3.72 EntityStateEVRStringType 66
4.3.73 EntityStateVersionType 66
4.3.74 EntityStateFileSetRevisionType 67
4.3.75 EntityIOSVersionType 67
4.3.76 EntityStateStringType 67
4.3.77 EntityStateRecordType 68
4.3.78 EntityStateFieldType 68
4.4 OVAL Variables Model 69
4.4.1 oval_variables 69
4.4.2 VariablesType 69
4.4.3 VariableType 70
4.5 OVAL System Characteristics Model 70

4.5.1 SystemInfoType 71
4.5.2 InterfacesType 71
4.5.3 InterfaceType 71
4.5.4 CollectedObjectsType 72
4.5.5 ObjectType 72
4.5.6 VariableValueType 73
4.5.7 ReferenceType 73
4.5.8 SystemDataType 73
4.5.9 ItemType 73
4.5.10 EntityAttributeGroup 74
4.5.11 FlagEnumeration 74
4.5.12 StatusEnumeration 75
4.5.13 EntityItemSimpleBaseType 75
4.5.14 EntityItemComplexBaseType 75
4.5.15 EntityItemIPAddressType 76
4.5.16 EntityItemIPAddressStringType 76
4.5.17 EntityItemAnySimpleType 76
4.5.18 EntityItemBinaryType 76
4.5.19 EntityItemBoolType 77
4.5.20 EntityItemFloatType 77
4.5.21 EntityItemIntType 77
4.5.22 EntityItemStringType 77
4.5.23 EntityItemRecordType 77
4.5.24 EntityItemFieldType 78
4.5.25 EntityItemVersionType 78
4.5.26 EntityItemFileSetRevisionType 78
4.5.27 EntityItemIOSVersionType 78
4.5.28 EntityItemEVRStringType 79
4.6 OVAL Results Model 79
4.6.1 DirectivesType 80
4.6.2 DefaultDirectivesType 81

4.6.3 ClassDirectivesType 81
4.6.4 DirectiveType 82
4.6.5 ResultsType 82
4.6.6 SystemType 82
4.6.7 DefinitionType 83
4.6.8 CriteriaType 84
4.6.9 CriterionType 84
4.6.10 ExtendDefinitionType 85
4.6.11 TestType 86
4.6.12 TestedItemType 88
4.6.13 TestedVariableType 88
4.6.14 ContentEnumeration 88
4.6.15 ResultEnumeration 89
4.7 OVAL Directives Model 89
5 Processing Model for the OVAL Language 90
5.1 Producing OVAL Definitions 91
5.1.1 Reuse of Definition, Test, Object, State, and Variable 92
5.1.2 Tracking Change 92
5.1.3 Metadata 92
5.1.4 Content Integrity and Authenticity 92
5.2 Producing OVAL System Characteristics 92
5.2.1 System Information 93
5.2.2 Collected Objects 93
5.2.3 Conveying System Data without OVAL Objects 94
5.2.4 Recording System Data and OVAL Items 94
5.2.5 Content Integrity and Authenticity 97
5.3 Producing OVAL Results 97
5.3.1 Definition Evaluation 97
5.3.2 Test Evaluation 99
5.3.3 OVAL Object Evaluation 103
5.3.4 OVAL State Evaluation 108

5.3.5 OVAL Variable Evaluation 110
5.3.6 Common Evaluation Concepts 117
int 122
ipv4_address 122
5.3.7 Masking Data 128
5.3.8 Entity Casting 128
6 XML Representation 129
6.1 Signature Support 130
6.2 XML Extensions 130
6.3 ElementMapType 130
6.4 Official OVAL Component Models 131
6.5 Use of xsi:nil 132
6.6 Validation Requirements 132
Appendix A – Extending the OVAL Language Data Model 133
OVAL Component Models 133
OVAL Definitions Model 133
OVAL System Characteristics Model 135
Extension Points within the OVAL Definitions Model 135
Generator Information 135
OVAL Definition Metadata 135
Extension Points within the OVAL System Characteristics Model 135
Generator Information 135
System Information 136
OVAL Results Model 136
Generator Information 136
Appendix B - OVAL Language Versioning Policy 137
Appendix C - OVAL Language Deprecation Policy 137
Appendix D - Regular Expression Support 138
Supported Regular Expression Syntax 138
Metacharacters 138
Greedy Quantifiers 138

Reluctant Quantifiers 139
Escape Sequences 139
Character Classes 139
Zero Width Assertions 139
Extensions 139
Version 8 Regular Expressions 139
Appendix E – Normative References 139
Appendix F - Change Log 141
Appendix G - Terms and Acronyms 142
Terms 142
Acronyms 143

1 Introduction

Information security is a function that consumes significant organizational resources, and is growing increasingly difficult to manage. One of the biggest problems is the lack of standardization between the sources of security information, and the tools that consume that information, as well as between the various tools themselves. Often, the exchange of security information is time critical, but is hampered by the variety of incompatible formats in which it is represented.

This lack of standardization gives rise to many challenges across the information security community. Once such challenge is the ability to obtain the information necessary to detect the presence of a vulnerability. Generally, security advisories are released for a specific issue as a text document and often do not contain all of the information necessary to determine if the vulnerability exists on a specific system or not. This leaves the IT Security Professional with the task of investigating all available sources regarding the vulnerability and then trying to piece together the details for detecting the issue.

The next challenge involves the need for vulnerability content teams to reverse-engineer security advisories such that they can develop tests for their vulnerability and remediation tools. Often times, the content teams are writing vulnerability content for software that they are not intimately familiar with meaning the methodology used to detect the presence of a vulnerability is based on the interpretation of an individual analyst. As a result, different approaches are taken for different tools when searching for the presence of a vulnerability which leads to conflicting results on the same system. Once again, the burden falls upon the IT Security Professional to deconflict the results by examining the individual approaches taken by each of the tools and, if possible, decide which is correct.

Another challenge for the IT Security Professional is the usability of security configuration information. For organizations publishing security configuration information, there are often multiple repositories of configuration information, multiple ways in which to manipulate that data, and in some cases, complex precedence relationships between the data. It is time-consuming and error-prone for the IT Security Professional to read a configuration document, interpret its meaning with respect to a specific configuration setting, and then apply that knowledge to an actual system to determine an answer.

Organizations cannot rely on a single tool to provide a complete view of the systems on their network. Multiple tools are needed and, if they are from different vendors, it is very likely that they will use different formats for representing data inhibiting interoperability. This requires the IT Security Professional to correlate the data produced by the tools in order to obtain a complete view of the systems on the network. It may also be necessary for the data to be manually converted into a format that is usable by another tool which can also be a tedious and error-prone process.

What the industry requires is a standardized method for representing the configuration state of computer system, comparing it against some known state, and expressing the results of that comparison. The representation of this information must easily facilitate its consumption by a software tool. The advantage of such a standard is that it will:

- Significantly shorten the time between the official announcement of an issue and the ability of a tool to check for it.
- Bring consistency and transparency to the results produced by security scanning tools.
- Assist in the exchange of information between security tools.
- Reduce the need for IT Security Professionals to learn the proprietary languages of each of their tools, and instead allow them to learn a single language that is understood by all the tools.

This document presents the OVAL Language as a standard that fulfills these needs and requirements.

1.1 The OVAL Language

The Open Vulnerability and Assessment Language (OVAL®) is an international, information security, community standard to promote open and publicly available security content, and to standardize the transfer of this information across the entire spectrum of security tools and services. The OVAL Language, developed by a broad spectrum of industry, academia, and government organizations from around the world, standardizes the three main steps of the assessment process: *OVAL System Characteristics* for representing the configuration information of systems for testing; *OVAL Definitions* for expressing a specific machine state; and *OVAL Results* for reporting the results of the assessment. By doing so, the three core components of the OVAL Language serve as the framework and vocabulary of the OVAL Language and provide:

- A simple and straightforward approach for determining if a vulnerability, software application, configuration issue, or patch exists on a given system.
- A standard format that outlines the necessary security-relevant configuration information and encodes the precise details of a specific issue.
- An open alternative to closed, proprietary, and replicated efforts.
- An effort that is supported by a community of security experts, system administrators, and software developers from industry, government, and academia.

All of which leads to a common and structured format that facilitates collaboration and information sharing among the information security community as well as interoperability among security tools.

1.2 Document Conventions

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*.[16]

The following font and font style conventions are used throughout the remainder of this document:

- The `Courier New` font is used for writing constructs in the OVAL Language Data Model. Example: `generator`
- The *'italic, with single quotes'* font is used for noting values for OVAL Language properties. Example: *'does not exist'*

This document uses the concept of namespaces[3] to logically group OVAL constructs throughout both the Data Model section of the document, as well as other parts of the specification. The format of these namespaces is `prefix:element`, where the prefix is the namespace component, and the element is the name of the qualified construct. The following table lists the namespaces used in this document:

Data Model	Namespace	Description	Example
OVAL Common	oval	The OVAL Common data model that captures all of the common constructs used in OVAL.	`oval:GeneratorType`
OVAL Definitions	oval-def	The OVAL Definitions data model that defines the core framework constructs for creating OVAL Definitions.	`oval-def:TestType`
OVAL Results	oval-res	The OVAL Results data model that captures all the constructs used to communicate assessment results.	`oval-res:ResultsType`
OVAL Variables	oval-var	The OVAL Variables data model, used to define all constructs used to create OVAL Variables.	`oval-var:VariableType`
OVAL Directives	oval-dir	The OVAL Directives data model, which defines the constructs used to create OVAL Directives.	`oval-dir:oval_directives`
OVAL System Characteristics	oval-sc	The OVAL System Characteristics data model, which defines the constructs used to capture the data collected on a target system.	`oval-sc:ItemType`
External	ext	This namespace is used to identify those constructs that are defined outside the OVAL Language.	`ext:Signature`

1.3 Document Structure

This document serves as the specification for the OVAL Language defining the use cases, requirements, data model, and processing model which is organized into the following sections:

- Section 1 – Introduction
- Section 2 – Use Cases for the OVAL Language
- Section 3 – Requirements for the OVAL Language
- Section 4 – Data Model for the OVAL Language

[3] Namespaces (computer science): http://en.wikipedia.org/wiki/Namespace_(computer_science)

- Section 5 – Processing Model for the OVAL Language
- Section 6 – XML Representation
- Appendix A – Extending the OVAL Language Data Model
- Appendix B – OVAL Language Versioning Policy
- Appendix C – OVAL Language Deprecation Policy
- Appendix D – Regular Expression Support
- Appendix E – References
- Appendix F – Change Log
- Appendix G – Terms and Acronyms

2 Use Cases for the OVAL Language

OVAL Use Cases define the intended best practice usage of the standard. The current set of supported OVAL Use Cases are described below including one or more detailed use case scenarios for each use case. Additional use cases will be documented as they emerge through the continued operational application of OVAL.

2.1 Security Advisory Distribution

Security advisories are published by vendors and security researchers as product vulnerabilities are discovered. Security advisories generally contain the information needed to detect the presence of the vulnerable product on a system. These advisories are leveraged by alerting services and vulnerability scanning products to raise awareness of the latest issues that might affect individuals and organizations using the vulnerable products. One acknowledged need within the security industry is for application and operating system vendors, and other authoritative organizations, to publish vulnerability information in a standard, machine-readable format. The benefit of this is two-fold. First, it provides scanning products with immediate access to actionable content that can be used to assess the security posture of a system. Second, it moves the authoring of the technical details of a vulnerability from the reverse engineering efforts of the implementing organization (e.g., scanner-product developer) to a more authoritative source: the developer of the vulnerable product.

Use Case Scenario: Publishing an Advisory

In this scenario, a software vendor receives a report of an undisclosed vulnerability along with exploit code from a member of the security community. The vendor examines the report and the exploit code and confirms that there is a vulnerability in their software. The vendor further investigates the vulnerability to determine what versions of the software are affected and on what platforms. The vendor reserves a Common Vulnerabilities and Exposures (CVE®) Identifier[4] for the vulnerability and creates a standardized check for the vulnerability in the form of an OVAL Definition. This new OVAL

[4] Common Vulnerabilities and Exposures (CVE): https://cve.mitre.org

Definition includes the list of affected platforms and products, a reference to the reserved CVE Identifier, and a description of the vulnerability. The software vendor adds tests to check for the vulnerable software on the relevant platforms. Once complete, the OVAL Definition is signed to ensure integrity and authenticity and tested to ensure that it accurately detects all known vulnerable versions of the software. Finally, the software vendor publishes a new security advisory for the vulnerability including the reserved CVE Identifier and the OVAL Definition that will detect the presence of the vulnerability.

Immediately after publication, organizations begin to download the security advisory's OVAL Definition, verify its signature to ensure that it was not modified in transit, and use it in their vulnerability scanning tool of choice to determine whether or not their systems are vulnerable.

2.2 Vulnerability Management

Vulnerability management is the process of identifying the vulnerabilities in a system and prioritizing them according to their severity. Currently, organizations that develop vulnerability management products need to employ a team of content developers. This team investigates vulnerabilities as they become known, gathering all of the available information for a given vulnerability, and running various tests against live systems to examine the parameters that indicate the presence of a vulnerability. Once a vulnerability is understood, this team develops a check that will indicate the presence of the vulnerability on a system for use in their product. The resulting checks are then distributed to vendor's customers so that they can assess their systems and take action based on the vulnerability management results. All of these tasks must be completed under a very strict time requirement and are repeated across nearly every organization that develops and offers a vulnerability management product.

For vulnerability management product vendors, having vulnerability information structured in a standard format allows them to quickly consume data from multiple sources. These vendors can share vulnerability checks with each other and collaborate on developing the best possible check for a given issue. If the initial security advisory includes a standardized check for the issue, these vendors can automatically consume that data. This will allow the vendor to refocus resources away from content generation to tasks that enhance the functionality of their product while distributing higher quality checks more quickly to their customers.

From the product customer's perspective, the primary requirement for having a standard content format is that it demystifies the vulnerability management process and provides them with the ability to do an apples-to-apples comparison of the products. When conducting product comparisons, given a specific set of definitions, each product tested should return the same result. If this is not the case, it is no longer a result of the products taking different approaches to detecting a vulnerability, and removes the burden from the customer to determine which product they think returns the most accurate results. The end result is that the customer can focus more on selecting a product with the features that best meet their needs, and less on the more difficult problem of which product does the correct job of detecting vulnerabilities. Lastly, having a well-documented, standard format provides users with the

information they need to be able to understand the details of an issue, and to determine how a specific product is conducting its business.

Use Case Scenario: Leveraging a Standardized Security Advisory

An operating system vendor releases a new set of security advisories for its platform as OVAL Definitions. A system administrator runs the organization's vulnerability management tool which retrieves the OVAL Definitions and verifies its signature. The vulnerability management tool then collects the attributes required to make an assertion about whether or not the system is in a vulnerable state and includes this information in the OVAL System Characteristics. Next, the vulnerability management tool evaluates the OVAL System Characteristics against the OVAL Definitions and expresses the findings in the OVAL Results.

Use Case Scenario: Collaborating on the Development of a Vulnerability Check

A new critical vulnerability is disclosed by an application vendor and the initial security advisory does not include an authoritative standardized check for the vulnerability. A vulnerability management product vendor quickly develops and distributes an OVAL Definition with a check for the presence of the vulnerability on the platforms the vendor supports for its customers. The vendor shares this new check with a forum of other vulnerability management vendors and industry experts in the form of an OVAL Definition. The OVAL Definition is extended by another vendor to include detailed checking information for additional platforms in order to make the vulnerability check complete for all known vulnerable platforms. The resulting OVAL Definition is again shared with the industry forum. A security expert participating in the forum notices that under some circumstances the OVAL Definition will detect the vulnerability when in fact it does not exist (a false positive). The security expert corrects this defect in the OVAL Definition and once again shares this information with the forum. The forum members have collaborated in developing a thorough, accurate, standardized check for the vulnerability and leveraged the resulting OVAL Definition in their products and services.

Use Case Scenario: Sharing Vulnerability Assessment Results

A vulnerability management product, using an OVAL Definition, detects the presence of a vulnerability on a system and generates the OVAL Results that record this finding. The OVAL Results are provided to the organization's security dashboard where it is processed. Due to the severity of the vulnerability and availability of a patch it is determined that the affected system must be patched. The OVAL Results are then provided as an input to the organization's patch management tool where the affected system is identified and the appropriate patch for the vulnerability is identified by its CVE Identifier and applied to the system. The system is no longer in a vulnerable state.

2.3 Patch Management

Patch management is the process of identifying the security issues and software updates that affect a system, applying the patches that resolve these issues, and verifying that the patches were successfully installed. Ensuring that systems are properly patched is a major concern among organizations as it's a leading cause for compromised systems. Patch management tools must have a detailed understanding of what it means for a given patch to have been properly installed on a system to ensure that systems

are properly patched. As a result, patch management vendors employ teams of analysts to reverse engineer patches and fully understand the impact of applying a given patch to a system. These analysts must develop and maintain checks for each patch their product supports.

For the patch management vendor community, having patch checking information structured in a standard format allows them to quickly consume data from multiple sources. These vendors can share patch checks with each other and collaborate on developing the best possible check for a given patch. If the patch is distributed with a standardized check for the patch these vendors can automatically consume that data. This will allow the vendor to refocus resources away from content generation to tasks that enhance the functionality of their product while distributing higher quality patch checks more quickly to their customers.

Use Case Scenario: Leveraging a Standardized Patch Check

An operating system vendor releases a new set of patches for its platform and includes standardized patch checks as OVAL Definitions. A system administrator runs the organization's patch management tool which retrieves the OVAL Definitions and verifies its signature. The patch management tool then collects the attributes required to make an assertion about whether or not the system needs to be patched and includes this information in the OVAL System Characteristics. Next, the patch management tool evaluates the OVAL System Characteristics against the OVAL Definitions and expresses the findings in the OVAL Results. The patch management tool examines the OVAL Results and determines that a patch should be installed. The patch is installed and the system is no longer vulnerable.

Use Case Scenario: Patching a Known Vulnerability

An organization's patch management tool examines the OVAL Results generated by a vulnerability management tool. The OVAL Results include summary information about all vulnerabilities that were checked and full details about the vulnerabilities that were found during a vulnerability assessment. The patch management tool uses the CVE Identifier associated with each OVAL Definition, included in the OVAL Results, to enumerate the available patches for the vulnerable software found on the system.

2.4 Configuration Management

The process of configuration management involves examining a machine's configuration state, comparing it against a known good or mandated configuration state, and reporting the results. There are a number of publicly available best practice configuration guides (e.g., the National Security Agency (NSA) Configuration Guides[5], or the National Institute of Standards and Technology (NIST) National Checklist Program[6]), and many more developed specifically for individual organizations. In many cases, these guides exist in paper form only, and it is up to the IT Staff to translate the document into

[5] The National Security Agency (NSA) Configuration Guides
http://www.nsa.gov/ia/guidance/security_configuration_guides/
[6] The National Institute of Standards and Technology (NIST) National Checklist Program
http://checklists.nist.gov/

something that can be applied and enforced on a consistent basis. There are also automated solutions available that can scan a system for compliance against a given configuration and offer tailoring capabilities to suit the specific needs of an organization. Unfortunately, these products often rely upon proprietary data formats, making it difficult to introduce new configuration policies to the product or move data from one product to another. Finally, as with some of the use cases above, divesting the language from the product provides the product vendor with a broad repository of content and allows them to focus on functionality and features.

Use Case Scenario: Configuration Guidance Distribution

An operating system vendor releases a new version of its operating system. Along with the initial release of the operating system, detailed secure configuration guidance is included. This guidance is intended to act as a baseline for the operating system's users to tailor for their own environments and security requirements. The operating system vendor includes the OVAL Definitions with this secure configuration guidance. Each OVAL Definition includes a reference to the relevant Common Configuration Enumeration (CCE™) Identifier[7] for correlation with other guidance and policy frameworks, and can be used to check that a system is compliant with the operating system vendor's recommendation for that configuration item. A system administrator runs the organization's configuration management tool and provides the OVAL Definitions as input. The configuration management tool then collects the attributes required to make an assertion about whether or not the system is complaint with the new operating system configuration guidance and includes this information in the OVAL System Characteristics. Next, the configuration management tool evaluates the OVAL System Characteristics against the OVAL Definitions and expresses the findings in the OVAL Results.

Use Case Scenario: Authoritative Policy Reuse

An organization has decided to develop a secure configuration guide for its desktop systems. Rather than create a new guide from scratch the organization leverages the secure configuration guidance recommended by the desktop operating system vendor. Since this policy was published in a standardized, machine readable format, with a collection of OVAL Definitions for checking compliance with the guide, the organization downloads the policy and tailors it to their environment. By way of example, the organization has a very strict password policy and needs to require a minimum password length of 14 characters on all desktop systems. Given that the operating system vendor recommended a minimum password length of 8 characters as a parameterized value, there is already an OVAL Definition in the published secure configuration for check minimum password length that can leveraged. The organization is able to simply tailor the minimum password length value setting it to 14 and reuse the rest of the checking logic in the OVAL Definition. The organization applies several other customizations to the policy by editing the published OVAL Definitions. Once completed, the organization inputs the new policy into its configuration management tool which begins monitoring all desktop systems for compliance with the policy.

---

[7] Common Configuration Enumeration (CCE): https://cce.mitre.org

Use Case Scenario: Compliance Reporting

An organization is required to be compliant with an official configuration baseline and report on the degree of compliance in order to meet the configuration baseline requirements of an industry body. This baseline has been expressed and published as a collection of OVAL Definitions and digitally signed by the authority that developed it. Free from needing to translate the baseline, the organization assesses its systems for compliance with the baseline using the organization's own configuration management tool and the OVAL Definitions. For each system, the configuration management tool collects the attributes required to make an assertion about the system and its compliance with the baseline. The tool then includes this information in the OVAL System Characteristics to represent the current state of the system. The configuration management tool evaluates the OVAL System Characteristics against the baseline defined by the OVAL Definitions and includes the differences between current system state and the desired configuration in the OVAL Results. The OVAL Results are then forwarded to the organization's compliance reporting tool where the results of the system's compliance to the baseline can be made available to the authorities to demonstrate compliance.

2.5 System Inventory

System inventory is the process of gathering a detailed listing of the applications installed on a given system. Large enterprises often have many versions of many applications running on wide variety operating systems. Organizations simply do not rely upon one vendor for all of the software that is running in their enterprises. This raises a considerable challenge when tracking software for licensing, vulnerability management, compliance, and other purposes. Application and operating system vendors need a standardized way to describe how to check for the presence of an application, and system inventory tool vendors need reach out to numerous application and operating system vendors for this information in order to accurately determine what is installed on a system. Currently, these system inventory tool vendors must develop their own checks for the presence of an application or operating system, which is often based on a best guess rather than authoritative knowledge of the system.

Use Case Scenario: Operating System Upgrade

An organization wants to upgrade its remaining systems to the newest version of an operating system. The organization tasks the system administrator with determining how many licenses need to be purchased. The system administrator downloads the OVAL Definitions that contain checks for all of the previous versions of the operating system along with references to Common Platform Enumeration (CPE™) Identifiers[8] that correspond to the specific platform associated with a check. The system administrator then runs the system inventory tools across the organization using the downloaded OVAL Definitions. The system inventory tool collects the attributes required to make an assertion about the software installed on the system and includes it in the OVAL System Characteristics as a snapshot of the observed state of the system. Next, the system inventory tool compares the OVAL System Characteristics to the OVAL Definitions, and records the findings in the OVAL Results. Finally, the system

[8] Common Platform Enumeration (CPE): https://cpe.mitre.org

inventory tool forwards the OVAL Results to the organization's reporting tool. The reporting tool leverages the CPE Identifiers in the OVAL Results to provide detailed information about the number and types of earlier versions of the operating system that were found allowing the system administrator to see how many licenses are required for the upgrade.

2.6 Malware Artifact Hunting

Incident coordination centers, organizations, and other members of the security community are actively discussing malware and sharing low-level system details that can be used to detect potentially compromised systems. These details are commonly shared as prose documents that require translation into actionable content prior to being used for system assessment. The need for a standard format to encode malware artifacts is widely acknowledged, and its use by incident coordination centers would be widespread.

Use Case Scenario: Detecting Compromised Systems

An organization discovers that one of their systems has been compromised by some malicious software. Immediately, the organization tasks their forensics team with investigating the infected systems. During the investigation, the forensics team notices that the infected system contains certain files that have been modified and that a previously undisclosed vulnerability was used to gain access to the system. Realizing that there is no publicly available check for this vulnerability, the forensics team creates an OVAL Definition with tests to check for the presence of the modified files found during the investigation. Depending on the forensic team's in-house tools the resulting OVAL Definition may be automatically generated from numerous possible sources including static analysis tool outputs or possibly hand written in the case where a more manual process was used to investigate the incident.

Once complete, the forensics team quickly pushes the new OVAL Definition to their host-based security system in order to determine how widespread the attack on their infrastructure really is, before their anti-virus vendor has published updated signatures. The host-based security system collects the attributes required to determine if a system has been compromised records this information in the OVAL System Characteristics to capture the system's current state. Next, the host-based security system compares the OVAL System Characteristics against the OVAL Definitions and records the differences in system state in the OVAL Results. Finally, the host-based security system forwards the OVAL Results to the organization's reporting tool where the OVAL Results are leveraged to provide detailed information about the number of systems that have been compromised. Thus, enabling the forensics team to quickly quarantine and remediate the issue.

Use Case Scenario: Sharing Checks for Threat Indicators

An organization has in-depth knowledge about a system compromise and exactly what artifacts reside on a system after compromise. The organization exports an OVAL Definition from its in-house malware database with tests to check for the presence of these artifacts, modified files, and new registry keys, shares the resulting OVAL Definition with their partners. The partners run the OVAL Definition on their systems. The partner organizations quickly learn that they too have compromised systems and begin collaborating with each other in developing a strategy to remediate the affected systems.

2.7 Network Access Control (NAC)

NAC is a technology that can be used to enforce endpoint configuration policies. Policy enforcement may result in a number of outcomes including, but not limited to, granting full network access, denying network access, or granting some form of limited access. Most NAC solutions allow for policy checking and enforcement both when an endpoint requests access to a network and on an ongoing basis to ensure continued policy conformance. NAC policies are often expressed in proprietary data formats, making it difficult to introduce new policies or share policies among NAC products. Finally, as with some of the use cases above, divesting the language from the NAC product provides the product vendor with a wider repository of content and allows them to focus more on functionality and features.

Use Case Scenario: Minimum Secure Configuration Baseline Enforcement

An organization has defined an endpoint configuration policy that requires a minimum secure configuration including the installation of antivirus software, activation and proper configuration of a host-based firewall, and current patch status for all major applications and the operating system. This policy is expressed as a collection of OVAL Definitions, where each OVAL Definition describes how to determine if an endpoint complies with a single requirement in the organizational policy. This set of OVAL Definitions is then distributed to the various NAC solutions that are in place within the enterprise allowing the organization to define the policy once in a standard format and distribute it to each NAC solution in place. The various NAC solutions begin enforcing endpoint policy compliance as described in the OVAL Definitions.

2.8 Auditing and Centralized Audit Validation

Audit validation is responsible for providing reports about the state of a machine at any given time in the past. There are two basic needs in this area. First and foremost is capturing machine configuration information at a level of granularity that allows an organization to monitor, track, and reconstruct the transition of a system's configuration from one state to another. The second need is that the data be stored in a standardized, data-centric format, thus ensuring that it is not bound to a specific product, which may or may not be available at the time it is necessary to review the data.

Use Case Scenario: Keeping Track of Change

An organization deploys a centralized audit validation system. When a new system joins the network, it is immediately scanned by the organization's vulnerability management, patch management, and configuration management tools based on the most up-to-date security advisories, patches, and policies. The resulting OVAL Results, associated with the scans, serve as the system's initial state in the centralized audit validation system. From then on, the system is scanned once a week to determine if anything has changed. If changes are discovered since the last update, the centralized audit validation system is updated with the latest OVAL Results. If at any point in time between the scheduled scans, the system falls into or out of compliance or a new patch has been installed, the centralized audit validation system is immediately updated with the new OVAL Results. The centralized audit validation system now contains multiple sets of OVAL Results documenting the various state changes to the system over time.

2.9 Security Information Management Systems (SIMS)

SIMS integrate the output of a variety of security, auditing, and configuration products, as well as their own agents, to build a comprehensive view of the security posture of an organization's network. The fewer data formats the SIM needs to understand the more flexible and powerful the product can be. Standardizing the data exchange formats between products greatly simplifies the interoperability requirements and provides the end users with a wider array of applications to choose from.

Use Case Scenario: Data Aggregation

A security information management system vendor utilizes the OVAL Results generated by vulnerability management tools, patch management tools, configuration management tools, and any other tool that produces OVAL Results as a primary format for data coming into their system. By doing so, the system can consume data from an entire range of tools in a straightforward manner without the need to translate different formats, of like data, into a single format before it can be analyzed.

3 Requirements for the OVAL Language

The following requirements have been developed based upon the goals of OVAL and the needs outlined in the use cases above. These requirements apply to the OVAL Language itself and establish the OVAL Language as the standardized framework for expressing the configuration state of computer systems. At the highest level are the Basic Requirements, which capture the essence of the goals and use cases. Each of these requirements is further expanded and refined into individual classes of requirements in the OVAL Definition Requirements, OVAL System Characteristics Requirements, and OVAL Results Requirements sections below.

3.1 Basic Requirements

The basic requirements listed in this section form the foundation of the OVAL Language and are further refined and expanded upon in the Detailed Requirements section of this document.

3.1.1 Expressing Expected Configuration State

- The language MUST be capable of expressing the desired configuration state of a system.

3.1.2 Representing Observed Configuration State

- The language MUST be capable of expressing the actual configuration state of a system.

3.1.3 Expressing Assessment Results

- The language MUST be capable of expressing where the actual system configuration differs from the desired configuration.

3.1.4 Content Integrity and Authenticity

The language MUST provide the ability to ensure the integrity and authenticity of all content written in the language.

3.2 Detailed Requirements

The detailed requirements expand upon the general requirements listed in the previous section.

3.2.1 General Content Requirements

These general requirements apply to all content written in the language.

- The language MUST require that all content specify the language version which it complies with.
- The language MUST require that all content specify when it was created.
- The language MUST allow content to contain information about the product name and version used to create the content.
- The language MUST allow content to contain additional information that is relevant to the creation of the document.

3.2.2 OVAL Definition Requirements

These requirements apply to OVAL Definitions and further refine the basic requirements listed above.

- All major components of the language MUST be reusable.
- Components of the language MUST have globally unique identifiers.
- Component identifiers MUST be structured to allow individual organizations to dynamically create identifiers without relying on an outside source and be ensured that global uniqueness is maintained.
- The language MUST allow for the exchange of collections of OVAL Definitions as a single unit of content.
- A collection of OVAL Definitions MUST contain all of the individual components used by each definition in the collection.
- The language MUST contain the structure and the means to create unbounded logical combinations of individual components.
- The language MUST provide the ability to negate logical statements.
- The language MUST allow tailoring of configuration values to meet organization or environment specific policies.
- The language MUST allow the current configuration of a system to be used as the basis of further identifying configuration items to examine.
- The language MUST provide a means to add an authoritative reference to an OVAL Definition.
- An OVAL Test SHOULD be capable of testing all of the configuration parameters retrieved from a corresponding system element.
- An OVAL Test SHOULD mirror, in name and structure, the configuration parameters retrieved from a system element.

3.2.3 OVAL System Characteristics Requirements

These requirements apply to OVAL System Characteristics and further refine the basic requirements listed above.

- OVAL System Characteristics MUST include sufficient asset identification information to uniquely identify the target system.
- OVAL System Characteristics MUST allow for any additional asset identification information about the target system to be represented.
- OVAL System Characteristics MUST provide an extensible model for representing items collected from a system.
- OVAL System Characteristics MUST provide information about whether a specific item exists or does not exist on a system.
- OVAL System Characteristics MUST allow for a clear linkage between the information that was found on a system and the information that was requested of the system.
- OVAL System Characteristics MUST allow for the exchange of system configuration information without any dependency on other OVAL content.
- OVAL System Characteristics MUST report the outcome of attempting to collect a specified set of system information.
- OVAL System Characteristics MUST provide a means for tools to convey additional information (error messages, informational messages, etc.) related to attempts to collect system information.

3.2.4 OVAL Results Requirements

These requirements apply to OVAL Results and further refine the basic requirements listed above.

- OVAL Results MUST contain information that uniquely identifies the specific system being reported on.
- OVAL Results MUST be capable of supporting different levels of detail in the reported results.
- OVAL Results MAY include the OVAL Definitions that were evaluated.
- OVAL Results MUST contain the analysis result for each OVAL Definition and each referenced OVAL component being reported upon.
- OVAL Results MAY include the System Characteristics that were collected.

4 Data Model for the OVAL Language

The core components of the OVAL Language Data Model standardize the three main steps of the assessment process, specifically:

1. Representing the configuration information of a system.
2. Analyzing the system for the presence of a particular machine state.
3. Reporting the results of the comparison between the specified machine state and the observed machine state.

The *OVAL Definitions Model* defines an extensible framework for making an assertion about a system that is 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. Using the OVAL Definitions Model various assertions can be made about a system including:

- Is the system in a vulnerable state?
- Is a specific patch installed or missing from the system?
- Is a certain piece of software installed on the system?
- Is the system in compliance with a particular set of configuration guidance?

The *OVAL Variables Data Model* defines all constructs used to create OVAL Variables and can be used, in conjunction with the OVAL Definitions Model, to externally specify values that can tailor content based on the OVAL Definitions Model at run-time. This tailoring can be applied to both the identification of which system data to examine and the description of the state of that system data.

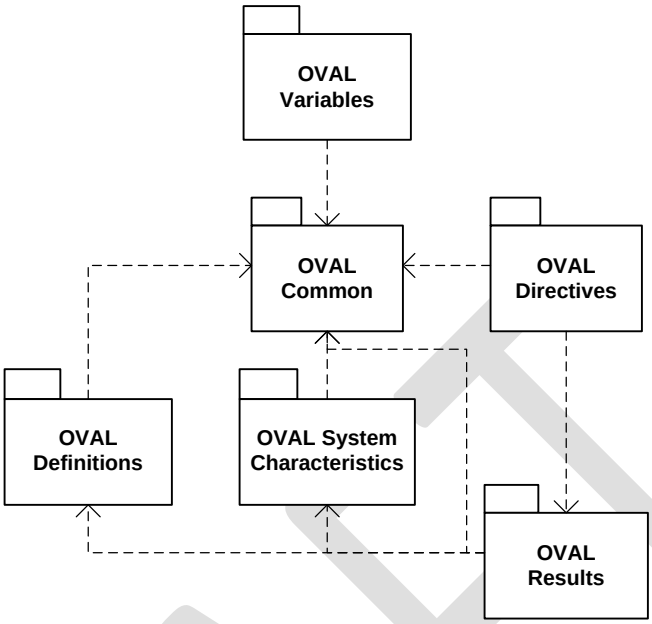
The *OVAL System Characteristics Model* provides a framework for representing low-level system configuration information that can be extended to support platform-specific constructs. The low-level system configuration information can include operating system properties, installed software, settings of installed software, operating system security settings, and other machine state. The low-level configuration information represented by the OVAL System Characteristics Model can be used to compare actual state against the expected machine state described by a set of OVAL Definitions.

The *OVAL Results Model* is used to report the results of an evaluation of a set of systems based upon a set of OVAL Definitions leveraging the OVAL System Characteristics. In this way, the OVAL Results Model provides detailed information about the set of assertions that were evaluated, the observed states of the evaluated systems, and the detailed results of the evaluation. This model enables applications to consume this data, interpret it, and take the necessary actions to report on the evaluation results or take other actions (for example, install patches, alter system configuration settings, and/or take external precautions to limit access to the affected systems). The OVAL Results Model can be tailored using the *OVAL Directives Model*, which defines the constructs used to create OVAL Directives, to include various levels of detail which allows for verbose detailed result information or streamlined result information based on a specific use case.

Lastly, many constructs and enumerations are reused throughout the different components in the OVAL Language Data Model. To facilitate reuse and avoid duplication, these common constructs and enumerations are represented in the *OVAL Common Model*.

The dependencies between the various components of the OVAL Language Data Model are depicted below.

Figure 4-1 Major Component Dependencies



4.1 Data Model Conventions

The following conventions are used throughout this data model section.

4.1.1 UML Diagrams

The Data Model makes use of Unified Modeling Language (UML)[9] diagrams where appropriate, to visually depict relationships for the OVAL Language constructs. Diagrams are included for any construct that inherits from other constructs or has a compositional relationship. The namespaces used in the diagrams map to those defined at the top of this document.

4.1.2 Property Table Notation

Throughout the data model, tables are used to describe each data type. Each property table will consist of a column of property names to identify the property, a type column to reflect the datatype of the property, a multiplicity column to reflect the allowed number of occurrences of the property, and a description column that will describe the property. Values in the type column are either primitive datatypes or other types defied in this document. These values will be cross referenced to the base definition of their types. Below is an example property table.

Table 4-1 Example Property Table

Property	Type	Multiplicity	Description
<PROPERTY NAME>	<DATA TYPE>	0..1	<DESCRIPTION OF THE PROPERTY AND ANY

[9] Unified Modeling Language - UML http://www.uml.org/

			USAGE REQUIREMENTS FOR THE PROPERTY>

4.1.3 Primitive Data Types

The following primitive datatypes are used in the OVAL Language.

- binary – Data of this type conforms to the World Wide Web Consortium (W3C) Recommendation for hex-encoded binary data [1].
- boolean – Data of this type conforms to the W3C Recommendation for boolean data [2].
- double – Data of this type conforms to the W3C Recommendation for double data [13].
- float – Data of this type conforms to the W3C Recommendation for float data [3].
- int – Data of this type conforms to the W3C Recommendation for integer data [4].
- string – Data of this type conforms to the W3C Recommendation for string data [6].
- unsigned int – Data of this type conforms to the W3C Recommendation for unsigned int data [15].
- URI – Data of this type conforms to the W3C Recommendation for anyURI data [14].
- DateTime – Data of this type represents a time value that conforms to the yyyy-mm-ddThh:mm:ss format.

4.2 OVAL Common Model

The OVAL Common Model contains definitions for constructs and enumerations that are used throughout the other core models in the OVAL Language Data Model both eliminating duplication and facilitating reuse.

4.2.1 GeneratorType

The `GeneratorType` provides a structure for recording information about how and when the OVAL Content was created, for what version of the OVAL Language it was created, and any additional information at the discretion of the content author.

Property	Type	Multiplicity	Description
product_name	string	0..1	Entity that generated the OVAL Content. This value SHOULD be expressed as a CPE Name.
product_version	string	0..1	Version of the entity that generated the OVAL Content.
schema_version	double	1	Version of the OVAL Language that the OVAL Content is expected to validate against.
timestamp	DateTime	1	The date and time of when the OVAL Content, in its entirety, was originally generated. This value is independent of the time at which any of the components of the OVAL Content were created.
extension_point	Any	0..*	An extension point that allows for the inclusion of any additional information associated with the generation of the OVAL Content.

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. For more information please see Appendix A – Extending the OVAL Language Data Model

4.2.2 MessageType

The `MessageType` construct is used to relay messages from tools at run-time. The decision of how to use these messages is left to the tool developer as an implementation detail based upon the context in which the message is used.

Property	Type	Multiplicity	Description
level	`MessageLevelEnumeration`	0..1	The level of the message. **Default Value:** *'info'*
message	string	1	The actual message relayed from the tool.

4.2.3 CheckEnumeration

The `CheckEnumeration` enumeration defines the acceptable values that can be used to determine the final result of an evaluation based on how many of the individual results that make up an evaluation are true. This enumeration is used in different contexts throughout the OVAL Language. See Section 5.3.6.1 Check Enumeration Evaluation, of the OVAL Language Processing Model, for more information on how this enumeration is used.

Enumeration Value	Description
all	The final result is *'true'* only if all of the individual results under consideration are *'true'*.
at least one	The final result is *'true'* only if one or more of the individual results under consideration are *'true'*.
none exist	**DEPRECATED (5.3)** In Version 5.3 of the OVAL Language, the checking of existence and state were separated into two distinct checks `CheckEnumeration` (state) and `ExistenceEnumeration` (existence). Since `CheckEnumeration` is now used to specify how many objects should satisfy a given state for a test to return true, and no longer used for specifying how many objects must exist for a test to return true, a value of *'none exist'* is no longer needed. The final result is *'true'* only if zero of the individual results under consideration are *'true'*.
none satisfy	The final result is *'true'* only if zero of the individual results under consideration are *'true'*.
only one	The final result is *'true'* only if one of the individual results under consideration is *'true'*.

4.2.4 ClassEnumeration

The `ClassEnumeration` defines the different classes of OVAL Definitions where each class specifies the overall intent of the OVAL Definition.

Enumeration Value	Description
compliance	This class describes OVAL Definitions that check to see if a system's state is compliant with a specific policy. An evaluation result of '*true*', for this class of OVAL Definitions, indicates that a system is compliant with the stated policy.
inventory	This class describes OVAL Definitions that check to see if a piece of software is installed on a system. An evaluation result of '*true*', for this class of OVAL Definitions, indicates that the specified software is installed on the system.
miscellaneous	This class describes OVAL Definitions that do not belong to any of the other defined classes.
patch	This class describes OVAL Definitions that check to see if a patch should be installed on a system. An evaluation result of *'true'*, for this class of OVAL Definitions, indicates that the specified patch should be installed on the system.
vulnerability	This class describes OVAL Definitions that check to see if the system is in a vulnerable state. An evaluation result of '*true*', for this class of OVAL Definitions, indicates that the system is in a vulnerable state.

4.2.5 SimpleDatatypeEnumeration

The `SimpleDatatypeEnumeration` defines the legal simple datatypes that are used to describe the values in the OVAL Language. Simple datatypes are those that are based upon a string representation without additional structure. Each value in the `SimpleDatatypeEnumeration` has an allowed set of operations listed in the table below. These operations are based upon the full list of operations which are defined in the `OperationEnumeration`.

Enumeration Value	Description
binary	Data of this type conforms to the W3C Recommendation for hex-encoded binary data [1]. Valid operations are: • equals • not equal
boolean	Data of this type conforms to the W3C Recommendation for boolean data [2]. Valid operations are: • equals • not equal
evr_string	Data of this type conforms to the format EPOCH:VERSION-RELEASE and comparisons involving this type MUST follow the algorithm of librpm's rpmvercmp() function. Valid operations are: • equals • not equal • greater than • greater than or equal

	• less than • less than or equal
fileset_revision	Data of this type conforms to the version string related to filesets in HP-UX. An example would be 'A.03.61.00'. Valid operations are: • equals • not equal • greater than • greater than or equal • less than • less than or equal
float	Data of this type conforms to the W3C Recommendation for float data [3]. Valid operations are: • equals • not equal • greater than • greater than or equal • less than • less than or equal
ios_version	Data of this type conforms to Cisco IOS Train strings. These are in essence version strings for IOS. Please refer to Cisco's IOS Reference Guide for information on how to compare different Trains as they follow a very specific pattern.[17] Valid operations are: • equals • not equal • greater than • greater than or equal • less than • less than or equal

int	Data of this type conforms to the W3C Recommendation for integer data [4]. Valid operations are: • equals • not equal • greater than • greater than or equal • less than • less than or equal • bitwise and • bitwise or
ipv4_address	The ipv4_address datatype represents IPv4 addresses and IPv4 address prefixes. Its value space consists of the set of ordered pairs of integers where the first element of each pair is in the range [0,2^32) (the representable range of a 32-bit unsigned int), and the second is in the range [0,32]. The first element is an address, and the second is a prefix length. The lexical space is dotted-quad CIDR-like notation ('a.b.c.d' where 'a', 'b', 'c', and 'd' are integers from 0-255), optionally followed by a slash ('/') and either a prefix length (an integer from 0-32) or a netmask represented in the dotted-quad notation described previously. Examples of legal values are '192.0.2.0', '192.0.2.0/32', and '192.0.2.0/255.255.255.255'. Additionally, leading zeros are permitted such that '192.0.2.0' is equal to '192.000.002.000'. If a prefix length is not specified, it is implicitly equal to 32. [19] Valid operations are: • equals • not equal • greater than • greater than or equal • less than • less than or equal • subset of • superset of
ipv6_address	The ipv6_address datatype represents IPv6 addresses and IPv6 address prefixes. Its value space consists of the set of ordered pairs of integers where the first element of each pair is in the range [0,2^128) (the representable range of a 128-bit unsigned int), and the second is in the range [0,128]. The first element is an address, and the second is a prefix length. The lexical space is CIDR notation given in IETF specification RFC 4291 for textual representations of IPv6 addresses and IPv6 address prefixes (see sections 2.2 and 2.3). If a prefix-length is not specified, it is implicitly equal to 128. [5]. Valid operations are:

	• equals • not equal • greater than • greater than or equal • less than • less than or equal • subset of • superset of
string	Data of this type conforms to the W3C Recommendation for string data [6]. Valid operations are: • equals • not equal • case insensitive equals • case insensitive not equal • pattern match
version	Data of this type represents a value that is a hierarchical list of non-negative integers separated by a single character delimiter. Any single non-number character may be used as a delimiter and the delimiter may vary between component of a given version string. Valid operations are: • equals • not equal • greater than • greater than or equal • less than • less than or equal

4.2.6 ComplexDatatypeEnumeration

The `ComplexDatatypeEnumeration` defines the complex datatypes that are supported the OVAL Language. These datatypes describe the values with some structure beyond simple string like content. O One simple example of a complex dataytype is an address. The address might be composed of a street, city, state, and zip code. These for field together comprise the complete address.

Each value in the `ComplexDatatypeEnumeration` has an allowed set of operations listed in the table below. These operations are based upon the full list of operations which are defined in the `OperationEnumeration`.

Enumeration Value	Description
record	Data of this type represents a collection of named fields and values. Valid operations are: • equals

4.2.7 DatatypeEnumeration

The `DatatypeEnumeration` defines the complete set of all valid datatypes. This set is created as the union of the `SimpleDatatypeEnumeration` and the `ComplexDatatypeEnumeration`. This type is provided for convenience when working with the OVAL Language.

4.2.8 ExistenceEnumeration

The `ExistenceEnumeration` defines the acceptable values that can be used to specify the expected number of components under consideration must exist.

Enumeration Value	Description
all_exist	The final existence result is *'true'* only if all of the components under consideration exist.
any_exist	The final existence result is *'true'* only if zero or more of the components under consideration exist.
at_least_one_exists	The final existence result is *'true'* only if one or more of the components under consideration exist.
none_exist	The final existence result is *'true'* only if zero of the components under consideration exist.
only_one_exists	The final existence result is *'true'* only if one of the components under consideration exist.

4.2.9 FamilyEnumeration

The `FamilyEnumeration` defines the high-level family that an operating system belongs to.

Enumeration Value	Description
catos	This value describes Cisco CatOS operating systems.
ios	This value describes Cisco IOS operating systems.
macos	This value describes Apple Mac OS operating systems.
pixos	This value describes Cisco PIX operating systems.
undefined	This value is reserved for operating systems where the high-level family is not available in the current enumeration.
unix	This value describes UNIX operating systems.
vmware_infrastructure	This value describes the VMWare Infrastructure.
windows	This value describes Microsoft Windows operating systems.

4.2.10 MessageLevelEnumeration

The `MessageLevelEnumeration` defines the different levels that can be associated with a message.

Enumeration Value	Description
debug	This level is reserved for messages that should only be displayed when the tool is run in verbose mode.
error	This level is reserved for messages where an error was encountered, but the tool could continue execution.

fatal	This level is reserved for messages where an error was encountered and the tool could not continue execution.
info	This level is reserved for messages that contain informational data.
warning	This level is reserved for messages that indicate that a problem may have occurred.

4.2.11 OperationEnumeration

The `OperationEnumeration` defines the acceptable operations in the OVAL Language. The precise meaning of an operation is dependent on the datatype of the values under consideration. See Section 5.3.6.3.1 Datatype and Operation Evaluation for additional information.

Enumeration Value	Description
equals	This operation evaluates to *'true'* if the actual value is equal to the stated value.
not equal	This operation evaluates to *'true'* if the actual value is not equal to the stated value.
case insensitive equals	This operation evaluates to *'true'* if the actual value is equal to the stated value when performing a case insensitive comparison.
case insensitive not equal	This operation evaluates to *'true'* if the actual value is not equal to the stated value when performing a case insensitive comparison.
greater than	This operation evaluates to *'true'* if the actual value is greater than the stated value.
less than	This operation evaluates to *'true'* if the actual value is less than the stated value.
greater than or equal	This operation evaluates to *'true'* if the actual value is greater than or equal to the stated value.
less than or equal	This operation evaluates to *'true'* if the actual value is less than or equal to the stated value.
bitwise and	This operation evaluates to *'true'* if the result of the BITWISE AND operation between the binary representation of the stated value and the actual value is equal to the binary representation of the stated value. This operation is used to determine if a specific bit in a value is set.
bitwise or	This operation evaluates to *'true'* if the result of the BITWISE OR operation between the binary representation of the stated value and the actual value is equal to the binary representation of the stated value. This operation is used to determine if a specific bit in a value is not set.
pattern match	This operation evaluates to *'true'* if the actual value matches the stated regular expression. The OVAL Language supports a common subset of the *Perl 5 Compatible Regular Expression Specification*. See Appendix D Regular Expression Support for more information about regular expression support in the OVAL Language.
subset of	This operation evaluates to *'true'* if the actual set is a subset of the stated set.
superset of	This operation evaluates to *'true'* if the actual set is a superset of the stated set.

4.2.12 OperatorEnumeration

The `OperatorEnumeration` defines the acceptable logical operators in the OVAL Language. See Section 5.3.6.2Operator Enumeration Evaluation for additional information.

Enumeration Value	Description
AND	This operator evaluates to *'true'* only if every argument is *'true'*.
ONE	This operator evaluates to *'true'* only if one argument is *'true'*.
OR	This operator evaluates to *'true'* only if one or more arguments are *'true'*.
XOR	This operator evaluates to *'true'* only if an odd number of arguments are *'true'*.

4.2.13 Definition, Test, Object, State, and Variable Identifiers

The identifiers used for OVAL Definitions, OVAL Tests, OVAL Objects, OVAL States, and OVAL Variables have a common structure based upon an Unified Resource Name (URN)[10] format with a type component that distinguishes one type of identifier from another. Each identifier has four components that are separated by a ':' and are represented in the following format:

<PREFIX>:<NAMESPACE>:<TYPE>:<ID>

These components are explained below:

- Prefix – The prefix is always "oval".
- Namespace – The namespace to which the identifier belongs.
- Type – For of the id. The allowed values are "def" for OVAL Definition, "tst" for OVAL Test, "obj" for OVAL Object, "ste" for OVAL State, and "var" for OVAL Variable.
- ID Value – The integer value of the identifier.

OVAL Definition, OVAL Test, OVAL Object, OVAL State, and OVAL Variable IDs are globally unique. Each ID MUST NOT be used more than once within the known body of OVAL Content.

The namespace portion of an ID SHOULD be represented as the reverse Domain Name System (DNS)[11] name of the organization that manages the content. Using a reverse DNS name provides a hint to the OVAL Community about the origin of the content and allows organizations to manage their own collections of IDs.

OVAL Definition, OVAL Test, OVAL Object, OVAL State, and OVAL Variable IDs SHOULD NOT contain any semantics. IDs are not intended to convey any meaning.

Once an OVAL Definition, OVAL Test, OVAL Object, OVAL State, or OVAL Variable IDs is assigned it SHOULD NOT be reused for any other OVAL Definition, OVAL Test, OVAL Object, OVAL State, or OVAL Variable.

---

[10] Unified Resource Name (URN): http://www.ietf.org/rfc/rfc3406.txt
[11] Domain Name System (DNS): http://tools.ietf.org/html/rfc1035

4.2.13.1 DefinitionIDPattern

The `DefinitionIDPattern` defines the URN format associated with OVAL Definition identifiers. All OVAL Definition identifiers MUST conform to the following regular expression:

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

4.2.13.2 ObjectIDPattern

The `ObjectIDPattern` defines the URN format associated with OVAL Object identifiers. All OVAL Object identifiers MUST conform to the following regular expression:

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

4.2.13.3 StateIDPattern

The `StateIDPattern` defines the URN format associated with OVAL State identifiers. All OVAL State identifiers MUST conform to the following regular expression:

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

4.2.13.4 TestIDPattern

The `TestIDPattern` defines the URN format associated with OVAL Test identifiers. All OVAL Test identifiers MUST conform to the following regular expression:

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

4.2.13.5 VariableIDPattern

The `VariableIDPattern` defines the URN format associated with OVAL Variable identifiers. All OVAL Variable identifiers MUST conform to the following regular expression:

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

4.2.14 ItemIDPattern

The `ItemIDPattern` defines the format associated with OVAL Item identifiers. All OVAL Item identifiers are unsigned integer values.

4.2.15 EmptyStringType

The `EmptyStringType` defines a string value with a maximum length of zero.

4.2.16 NonEmptyStringType

The `NonEmptyStringType` defines a string value with a length greater than zero.

4.2.17 Any

The `Any` datatype represents an abstraction that serves as the basis for other user defined datatypes. This `Any` datatype does not constrain its data in anyway. This type is used to allow for extension with the OVAL Language.

4.2.18 Signature

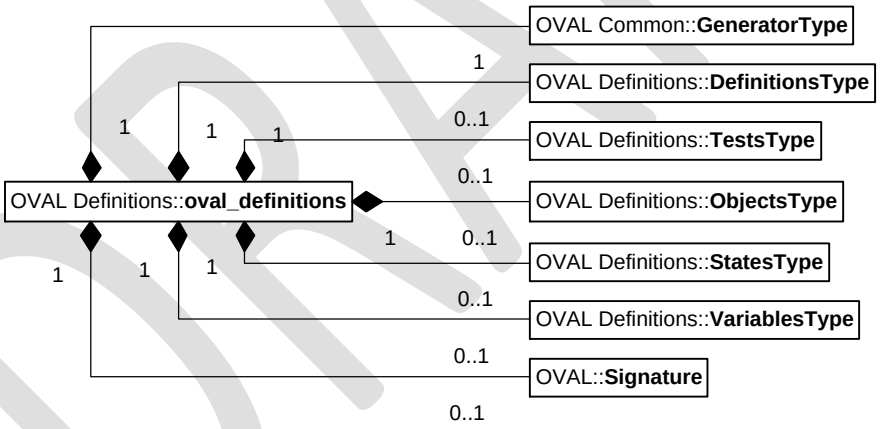
The `Signature` type provides a structure for applying a digital signature to OVAL Content. Any binding or representation of the OVAL Language MUST specify the format and structure of this type. This type is defined in an external namespace and when referenced in this document will be prefix with the external namespace alias as follows, `ext:Signature`. See Section 6.1 for more information on how signatures are used in the XML binding of OVAL.

4.3 OVAL Definitions Model

The OVAL Definitions Model provides a way to describe assertions about a system state. It combines the identification of required assessment data and the associated expected state of the data.

4.3.1 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	Multiplicity	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 States.
variables	VariablesType	0..1	Container for OVAL Variables.
signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the content.

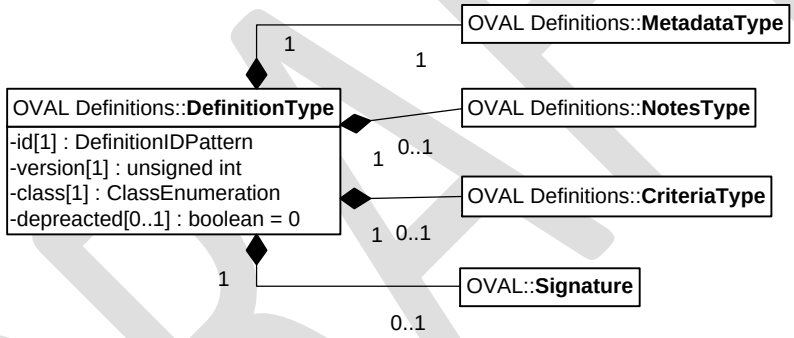
4.3.2 DefinitionsType

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

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

4.3.3 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.

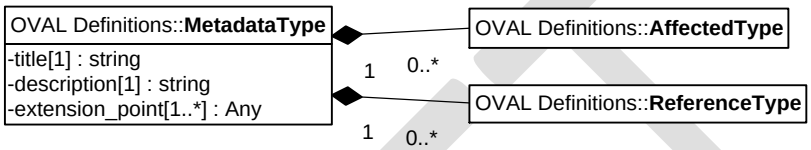


Property	Type	Multiplicity	Description
id	oval:DefinitionIDPattern	1	The globally unique identifier of the OVAL Definition.
version	unsigned integer	1	The version of the OVAL Definition.
class	oval:ClassEnumeration	1	The class of the OVAL Definition.
deprecated	boolean	0..1	Whether or not the OVAL Definition 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.

4.3.4 MetadataType

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



Property	Type	Multiplicity	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.

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. For more information on making use of this extension point see Appendix A – Extending the OVAL Language Data Model.

4.3.5 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	Multiplicity	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 names.
product	string	0..*	The name identifying a specific software product. Convention is to spell out names.

4.3.6 ReferenceType

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

Property	Type	Multiplicity	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.

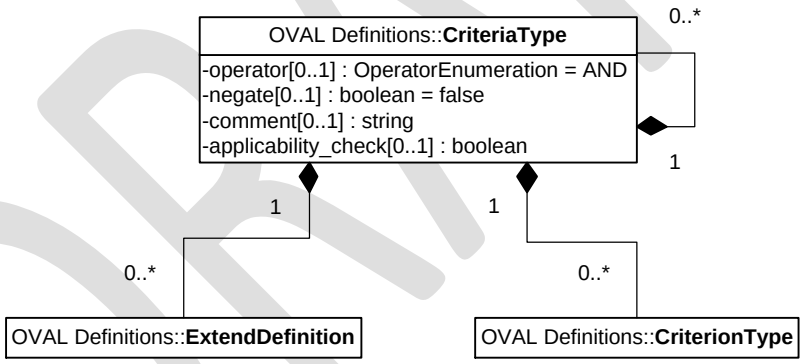
4.3.7 NotesType

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	Multiplicity	Description
note	string	1..*	One or more text notes.

4.3.8 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 `CriteriaType`s into one logical statement.

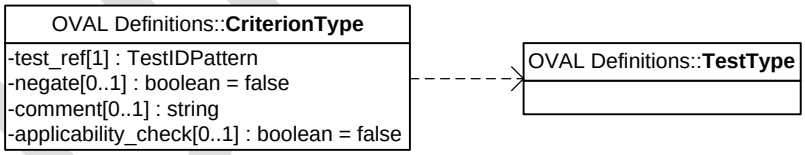


Property	Type	Multiplicity	Description
operator	oval:OperatorEnumeration	0..1	The logical operator that is used to combine the individual results of the logical statements defined by the `criteria`, `criterion`, and `extend_definition` properties. **Default Value:** *'AND'*
negate	boolean	0..1	Specifies 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'*.

4.3.9 CriterionType

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

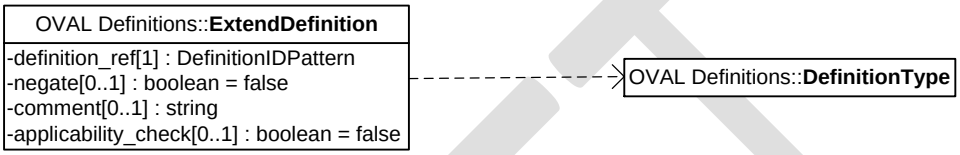


Property	Type	Multiplicity	Description
test_ref	oval:TestIDPattern	1	The globally unique identifier of an OVAL Test contained in the OVAL Definitions.
negate	boolean	0..1	Specifies whether or not the evaluation result of the OVAL Test, referenced by the `test_ref` property should be negated. **Default 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'*.

4.3.10 ExtendDefinitionType

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



Property	Type	Multiplicity	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'*.

4.3.11 TestsType

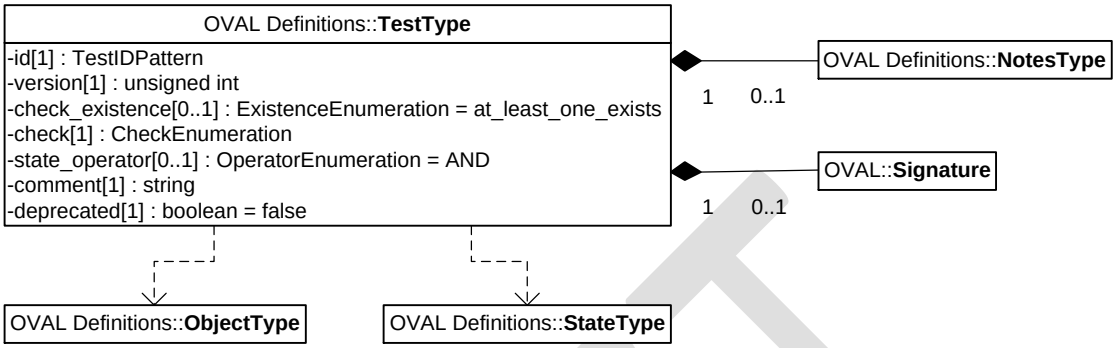
The `TestsType` provides a container for one or more OVAL Tests.

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

4.3.12 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, as described in the section on extending the Language

(Appendix A – Extending the OVAL Language Data Model). 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	Multiplicity	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.

4.3.13 ObjectRefType

The `ObjectRefType` points to an existing OVAL Object.

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

4.3.14 StateRefType

The `StateRefType` points to an existing OVAL State.

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

4.3.15 ObjectsType

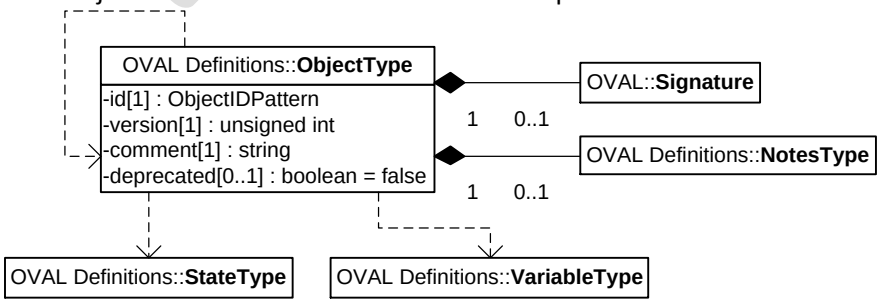
The `ObjectsType` provides a container for one or more OVAL Objects.

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

4.3.16 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.

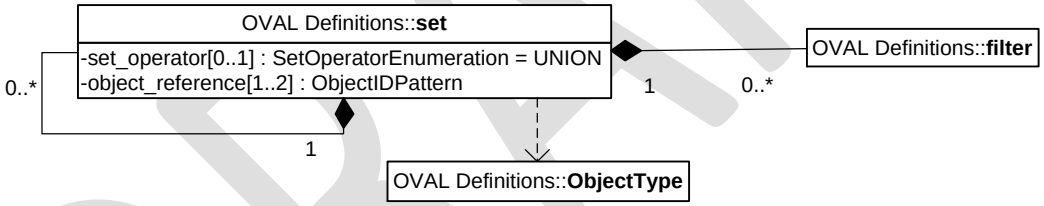
A concrete OVAL Object may define a set of 0 or more OVAL Behaviors. OVAL Behaviors define an action that can further specify the set of OVAL Items that match an OVAL Object. OVAL Behaviors may depend on other OVAL Behaviors or may be independent of other OVAL Behaviors. In addition, OVAL Behaviors are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



Property	Type	Multiplicity	Description
id	oval:ObjectIDPattern	1	The 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.

4.3.17 set

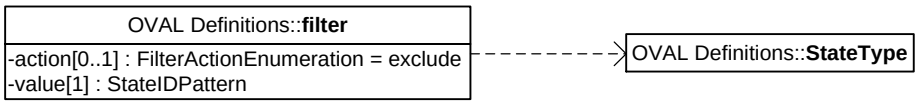
The `set` construct enables the expression of complex OVAL Objects that are the result of logically combining and filtering the OVAL Items that are identified by one or more other OVAL Objects. A `set` can consist of either one or two nested `sets` or one or two references to other OVAL Objects and a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



Property	Type	Multiplicity	Description
set_operator	SetOperatorEnumeration	0..1	Specifies the set operation to use when combining subsets. **Default Value:** *'UNION'*
set	set	0..2	Allows 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 the combined data.

4.3.18 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	Multiplicity	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.

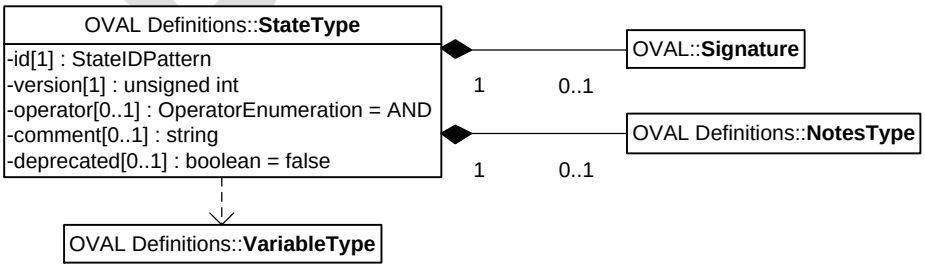
4.3.19 StatesType

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

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

4.3.20 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, as described in the section on extending the Language (Appendix A – Extending the OVAL Language Data Model). 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	Multiplicity	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 State 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.

4.3.21 VariablesType

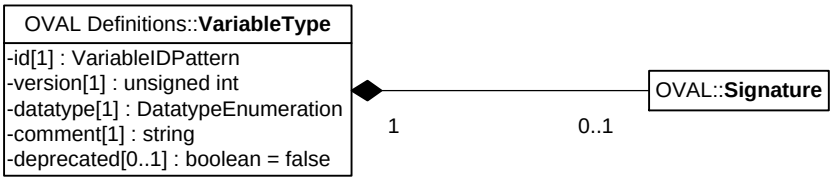
The `VariablesType` provides a container for one or more OVAL Variables.

Property	Type	Multiplicity	Description
variable	VariableType	1..*	A collection of OVAL Variables.

4.3.22 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 Section 5 Processing. All OVAL Variable values MUST conform to the datatype specified by the `datatype` property.

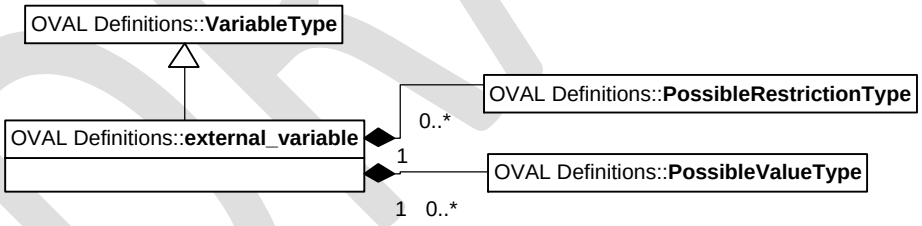


Property	Type	Multiplicity	Description
id	oval:VariableIDPattern	1	The unique identifier of an OVAL Variable contained in the OVAL Definitions
version	unsigned int	1	The version of the globally unique OVAL Variable referenced by the `id` property.
datatype	oval:SimpleDatatypeEnumeration	1	The datatype 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.

4.3.23 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	Multiplicity	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.

4.3.24 PossibleValueType

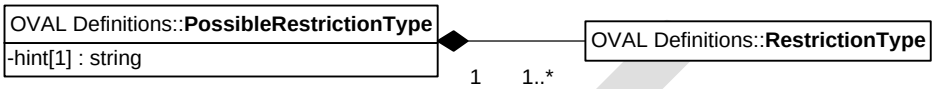
The `PossibleValueType` provides a way to explicitly state an acceptable value for an external variable.

Property	Type	Multiplicity	Description
hint	string	1	A short description that describes the allowed value.

value	string	1	An acceptable value for the external variable.

4.3.25 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 Section 5.3.9.2 Operator Enumeration Evaluation for more information on how to combine the individual results.



Property	Type	Multiplicity	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.

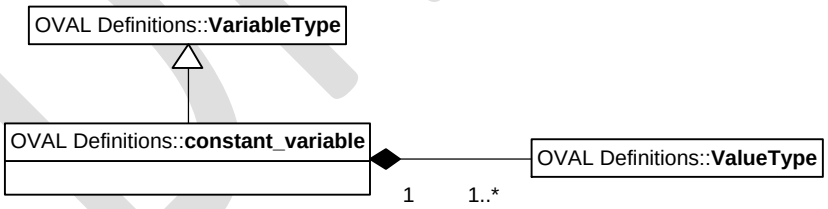
4.3.26 RestrictionType

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

Property	Type	Multiplicity	Description
operation	OperationEnumeration	1	The operation to be applied to the restriction.
value	string	1	An acceptable value for the external variable.

4.3.27 constant_variable

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



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

4.3.28 ValueType

The `ValueType` element defines a variable value.

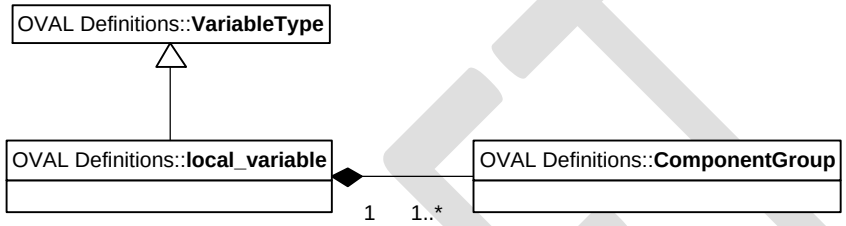
Property	Type	Multiplicity	Description
value	string	0..*	Allows any simple type to be used as a

			value. If no value is specified the value is considered to be the empty string.

4.3.29 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.

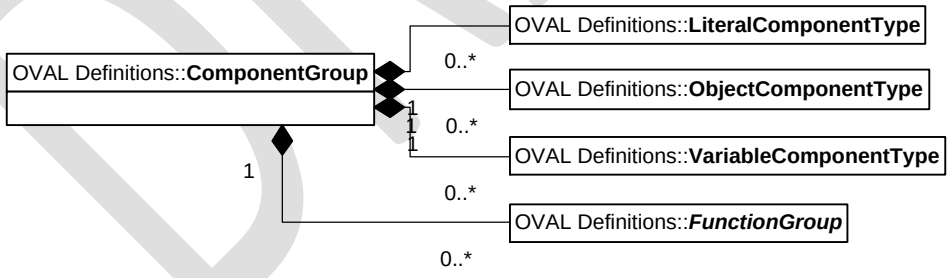
A `local_variable` can be constructed from a single component or via complex functions to manipulate the referenced components.



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

4.3.30 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	Multiplicity	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.

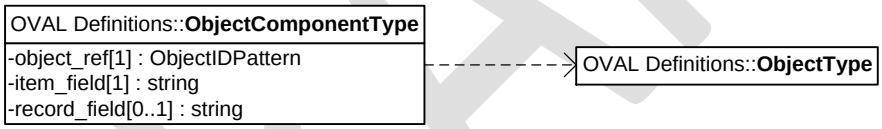
4.3.31 LiteralComponentType

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

Property	Type	Multiplicity	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.

4.3.32 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.

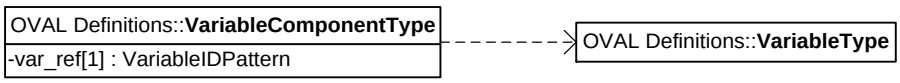


Property	Type	Multiplicity	Description
object_ref	oval:ObjectIDPattern	1	Specifies the Identifier for the OVAL Object to which the component refers.
item_field	oval:NonEmptyStringType	1	The name of the OVAL Item Entity to use for the value(s) of the OVAL Variable.
record_field	oval:NonEmptyStringType	0..1	Allows the retrieval of a specified OVAL field to be retrieved from an OVAL Item Entity that has a datatype of *'record'*.

4.3.33 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.

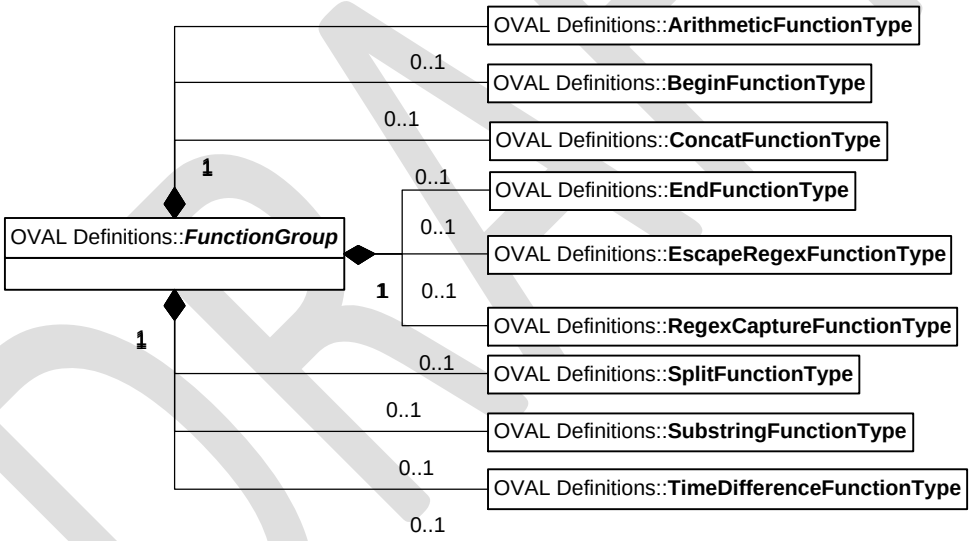
A variable component is a component that resolves to the value(s) associated with the referenced OVAL Variable.



Property	Type	Multiplicity	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

4.3.34 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.



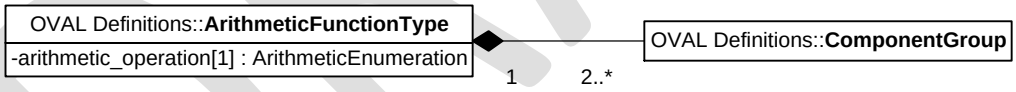
Property	Type	Multiplicity	Description
arithmetic	ArithmeticFunctionType	0..1	A function for performing basic math on numbers.
begin	BeginFunctionType	0..1	A function that ensures that a collected string starts with a specified string.
concat	ConcatFunctionType	0..1	A function that combines multiple strings.
count	CountFunctionType	0..1	A function that counts returns the count of all of the values represented by the components.
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 delimiter.
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.

4.3.35 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[12] of the values.

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



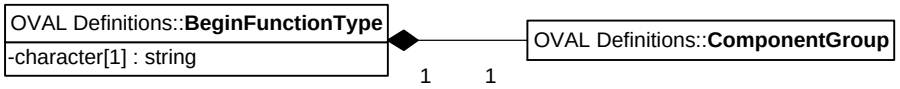
Property	Type	Multiplicity	Description
arithmetic_operation	ArithmeticEnumeration	1	The operation to perform.
values	ComponentGroup	2..*	Any type from the `ComponentGroup`.

4.3.36 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

[12] Cartesian Product http://en.wikipedia.org/wiki/Cartesian_product

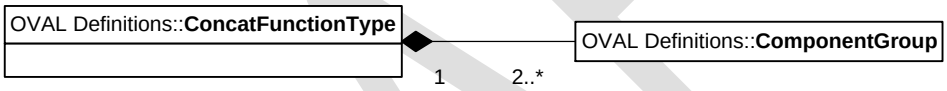
set of characters from the `character` property to the string. Otherwise, the string value will remain unchanged.



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

4.3.37 ConcatFunctionType

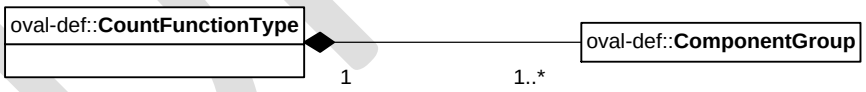
The `ConcatFunctionType` defines a function that concatenates the values specified together into a single string value. This function combines the values of two or more sub-components into a single string value. The function combines the sub-component values in the order that they are specified. That 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	Multiplicity	Description
values	ComponentGroup	2..*	Any type from the `ComponentGroup`.

4.3.38 CountFunctionType

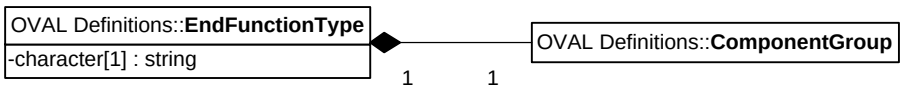
The `CountFunctionType` defines a function that counts the values represented by one or more components as an integer. This function determines the total number of values referenced by all of the specified sub-components.



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

4.3.39 EndFunctionType

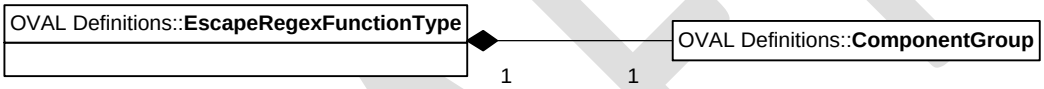
The `EndFunctionType` defines a function that ensures that the specified values end with a specified character or string. This function operates on a single sub-component of datatype string and ensures that the specified value(s) end with the characters specified in the `character` property. When a value 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	Multiplicity	Description
character	string	1	The character or string to use for the function.
value	ComponentGroup	1	Any type from the `ComponentGroup`.

4.3.40 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 Appendix D Regular Expression Support for more information on the subset of the Perl 5 regular expression syntax that is supported in the OVAL Language.



Property	Type	Multiplicity	Description
value	ComponentGroup	1	Any type from the `ComponentGroup`.

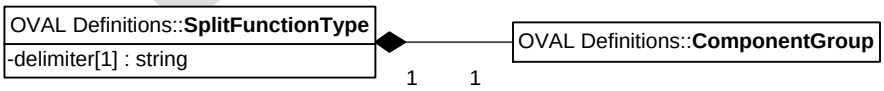
4.3.41 SplitFunctionType

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.

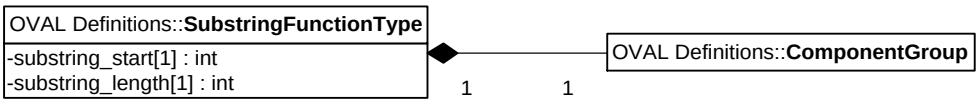
If the delimiter is not found in the subject string, the entire subject string will be included in the result.



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

4.3.42 SubstringFunctionType

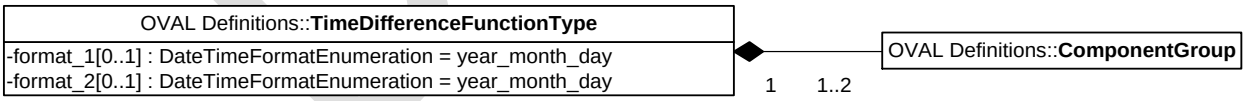
The `SubstringFunctionType` defines a function that takes a string value and produces a value that contains a portion of the original string.



Property	Type	Multiplicity	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`.

4.3.43 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	Multiplicity	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`.

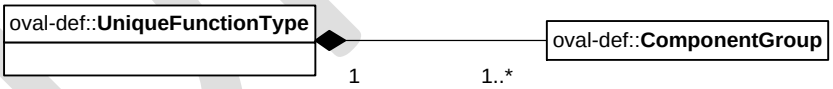
If a sub-component value does not conform to the format specified in the `DateTimeFormatEnumeration` an error MUST be reported.

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`.

DateTimeFormatEnumeration Value	Datatype
year_month_day	string
month_day_year	string
day_month_year	string
win_filetime	int
seconds_since_epoch	int

4.3.44 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. See `oval:DatatypeEnumeration` in Section 5.2.4.5.3 Datatype for more information on comparing two string values.



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

4.3.45 RegexCaptureFunctionType

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

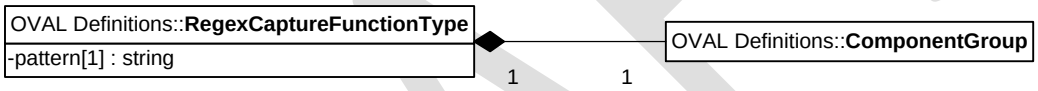
The pattern property specifies a regular expression, which SHOULD contain a single capturing sub-pattern (using parentheses). If the regular expression contains multiple capturing sub-patterns, only the first capture is used. If there are no capturing sub-patterns, the result for each target string MUST 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 Appendix D Regular Expression Support for more information on the subset of the Perl 5 regular expression syntax that is supported in the OVAL Language.

For more information about supported regular expressions, see the common subset of the Perl 5's regular expression syntax that the OVAL Language supports in Appendix D - Regular Expression Support.



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

4.3.46 ArithmeticEnumeration

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

Enumeration Value	Description
add	Indicates addition.
multiply	Indicates multiplication.

4.3.47 DateTimeFormatEnumeration

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

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

	• yyyy/mm/dd • yyyy-mm-dd hh:mm:ss • yyyy-mm-dd
month_day_year	This value indicates a format that follows the following patterns: • mm/dd/yyyy hh:mm:ss • mm/dd/yyyy • mm-dd-yyyy hh:mm:ss • mm-dd-yyyy • NameOfMonth, dd yyyy hh:mm:ss • NameOfMonth, dd yyyy • AbreviatedNameOfMonth, dd yyyy hh:mm:ss • AbreviatedNameOfMonth, dd yyyy
day_month_year	This value indicates a format that follows the following patterns: • dd/mm/yyyy hh:mm:ss • dd/mm/yyyy • dd-mm-yyyy hh:mm:ss • dd-mm-yyyy
win_filetime	This value indicates a date-time that follows the windows file time format[20].
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.

4.3.48 FilterActionEnumeration

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

Enumeration 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.

4.3.49 SetOperatorEnumeration

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

Enumeration 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.

4.3.50 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	Multiplicity	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.

4.3.51 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	Multiplicity	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.

4.3.52 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	Multiplicity	Description
attributes	EntityAttributeGroup	1	The standard attributes available to all entities.

4.3.53 EntityObjectIPAddressType

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

Property	Type	Multiplicity	Description

datatype	oval:SimpleDatatypeEnumeration	1	Possible values: • *'ipv4_address'* • *'ipv6_address'* Also allows an empty string value.

4.3.54 EntityObjectIPAddressStringType

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

Property	Type	Multiplicity	Description
datatype	oval:SimpleDatatypeEnumeration	1	Possible values: • *'ipv4_address'* • *'ipv6_address'* • *'string'* Also allows an empty string value.

4.3.55 EntityObjectAnySimpleType

The `EntityObjectAnySimpleType` extends the `EntitySimpleBaseType` and describes any simple data.

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

4.3.56 EntityObjectBinaryType

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

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

4.3.57 EntityObjectBoolType

The `EntityObjectBoolType` extends the `EntitySimpleBaseType` and describes any simple Boolean data.

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

4.3.58 EntityObjectFloatType

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

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

4.3.59 EntityObjectIntType

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

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

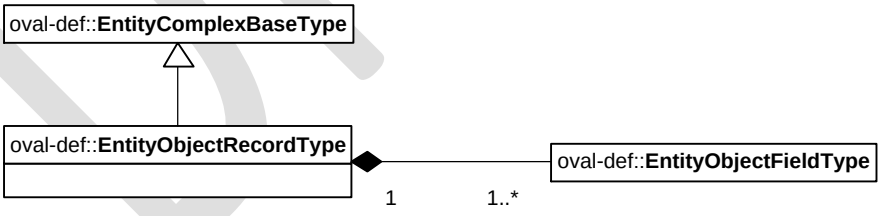
4.3.60 EntityObjectStringType

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

Property	Type	Multiplicity	Description
datatype	oval:SimpleDatatypeEnumeration	0..1	This value is fixed as *'string'*.

4.3.61 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	Multiplicity	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.

4.3.62 EntityObjectFieldType

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

Property	Type	Multiplicity	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.

4.3.63 EntityStateSimpleBaseType

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

Property	Type	Multiplicity	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.

4.3.64 EntityStateComplexBaseType

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

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

4.3.65 EntityStateIPAddressType

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

Property	Type	Multiplicity	Description
datatype	oval:SimpleDatatypeEnumeration	1	Possible values: • *'ipv4_address'* • *'ipv6_address'* Also allows an empty string value.

4.3.66 EntityStateIPAddressStringType

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

Property	Type	Multiplicity	Description
datatype	oval:SimpleDatatypeEnumeration	1	Possible values: • *'ipv4_address'* • *'ipv6_address'* • *'string'* Also allows an empty string value.

4.3.67 EntityStateAnySimpleType

The `EntityStateAnySimpleType` extends the `EntityStateSimpleBaseType` and describes any simple data.

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

4.3.68 EntityStateBinaryType

The `EntityStateAnyBinaryType` extends the `EntityStateSimpleBaseType` and describes any simple binary data.

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

4.3.69 EntityStateBoolType

The `EntityStateBoolType` extends the `EntityStateSimpleBaseType` and describes any simple Boolean data.

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

4.3.70 EntityStateFloatType

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

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

4.3.71 EntityStateIntType

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

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

4.3.72 EntityStateEVRStringType

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

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

4.3.73 EntityStateVersionType

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

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

4.3.74 EntityStateFileSetRevisionType

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

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

4.3.75 EntityIOSVersionType

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

Property	Type	Multiplicity	Description
datatype	oval:SimpleDatatypeEnumeration	1	Possible values: • *'ios_version'* • *'string'* The string type is an option in order to allow use of regular expressions. Also allows an empty string value.

4.3.76 EntityStateStringType

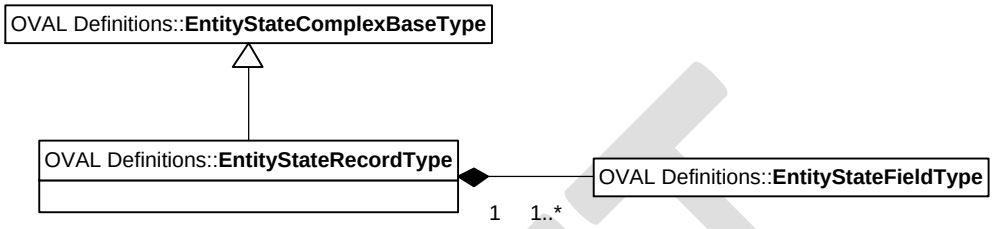
The `EntityStateStringType` extends the `EntityStateSimpleBaseType` and describes any simple string data.

Property	Type	Multiplicity	Description
datatype	oval:SimpleDatatypeEnumeration	0..1	This value is fixed as *'string'*.

			Also allows an empty string value.

4.3.77 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 Structured Query Language (SQL) statements and similar data.



Property	Type	Multiplicity	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.

4.3.78 EntityStateFieldType

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

Property	Type	Multiplicity	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.

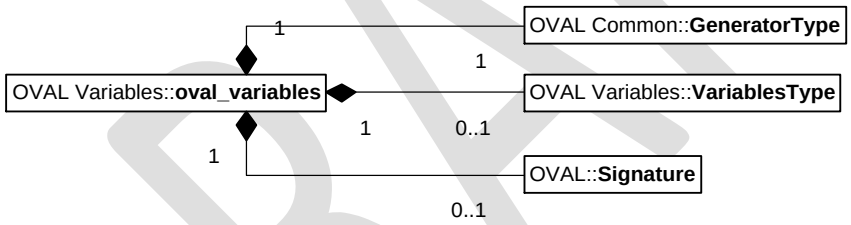
entity_check	oval:CheckEnumeration	0..1	Directs how to evaluate state for the entity. **Default Value:** *'all'*
value	string	0..1	The value of the field. An empty string value MUST be used when referencing an OVAL Variable.

4.4 OVAL Variables Model

The OVAL Variables Model contains constructs that allow for the specification of values for `external_variables` defined in content that was created using the OVAL Definitions Model. The OVAL Variables Model serves as a useful mechanism for parameterizing content based on the OVAL Definitions Model.

4.4.1 oval_variables

The `oval_variables` type defines the base structure in the OVAL Variables Model for representing a collection of OVAL Variables and their associated values. This container type adds metadata about the origin of the content and allows for a signature.



Property	Type	Multiplicity	Description
generator	oval:GeneratorType	1	Information regarding the generation of the OVAL Variables content. The `timestamp` property of the `generator` MUST represent the time at which the `oval_variables` was created.
variables	VariablesType	1	The variables defined in the OVAL Variables content.
signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the OVAL Variables content.

4.4.2 VariablesType

The `VariablesType` construct is a container for one or more OVAL Variables.

Property	Type	Multiplicity	Description

variable	VariableType	1..*	A collection of OVAL Variables.

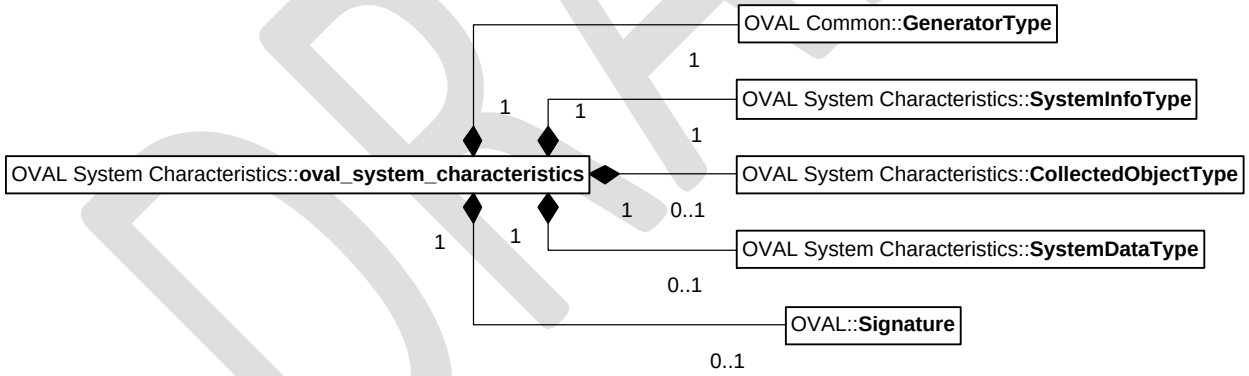
4.4.3 VariableType

The `VariableType` defines a variable in the OVAL Variables Model that corresponds to an instance of an external variable in content based on the OVAL Definitions Model.

Property	Type	Multiplicity	Description
id	oval:VariableIDPattern	1	The unique identifier of an external variable.
datatype	oval:SimpleDatatypeEnumeration	1	The datatype of the value(s) in the variable.
comment	string	1	The documentation associated with the variable instance.
value	string	1..*	The value(s) associated with the variable.

4.5 OVAL System Characteristics Model

The OVAL System Characteristics Model is used to represent low-level, system settings that describe the current state of a system. The OVAL System Characteristics Model serves as a basis for extension to create platform-specific, low-level configuration information models.

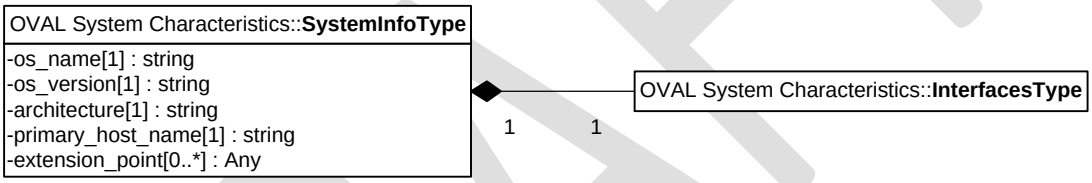


Property	Type	Multiplicity	Description
generator	oval:GeneratorType	1	Information regarding the generation of the OVAL System Characteristics. The `timestamp` property of the `generator` MUST represent the time at which the system state information was collected.
system_info	SystemInfoType	0..*	Information used to identify the

			system under test.
collected_objects	CollectedObjectsType	0..1	Contains the mapping between OVAL Objects defined in the OVAL Definitions and the OVAL Items that were collected from the system under test.
system_data	SystemDataType	0..1	Contains the OVAL Items that were collected from the system under test.
signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the OVAL System Characteristics content.

4.5.1 SystemInfoType

The `SystemInfoType` defines the basic identifying information associated with the system under test.



Property	Type	Multiplicity	Description
os_name	string	1	The operating system running on the system under test.
os_version	string	1	The version of the operating system running on the system under test.
architecture	string	1	The hardware architecture type of the system under test.
primary_host_name	string	1	The primary host name of the system under test.
interfaces	InterfaceType	0..*	The network interface(s) present on the system under test.
extension_point	Any	0..*	An extension point that allows for the inclusion of any additional identifying information associated with the system under test.

4.5.2 InterfacesType

The `InterfacesType` provides a container for zero or more interfaces.

Property	Type	Multiplicity	Description
interface	InterfaceType	0..*	One or more interfaces.

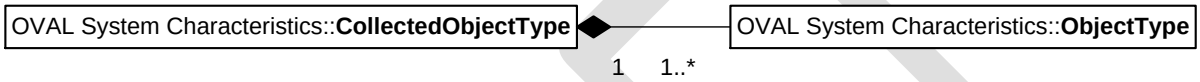
4.5.3 InterfaceType

The `InterfaceType` defines the information associated with a network interface on the system under test. This information may help to identify a specific system on a network.

Property	Type	Multiplicity	Description
interface_name	string	1	The name of the interface.
ip_address	string	1	The Internet Protocol (IP) address of the interface.
mac_address	string	1	The Media Access Control (MAC) address of the interface. MAC addresses MUST be formatted according to IEEE 802-2001 Section 9.2.1 [7].

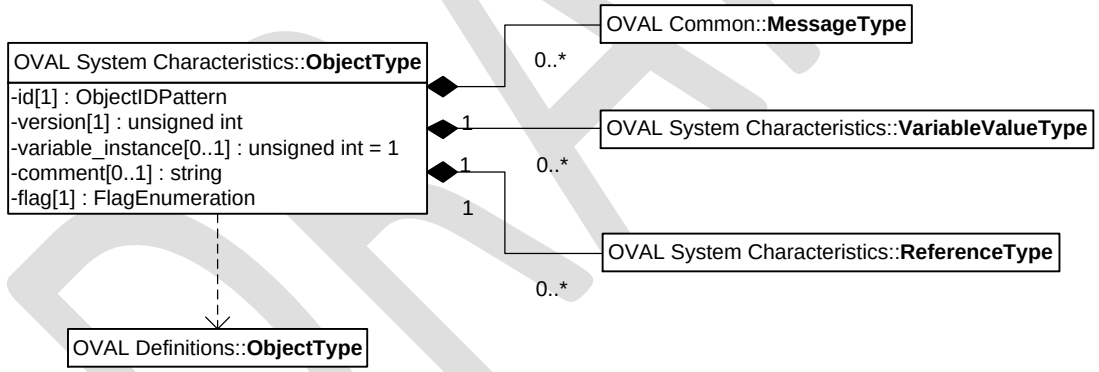
4.5.4 CollectedObjectsType

The `CollectedObjectType` is a container for one or more objects of type `ObjectType` that were used for data collection on the system under test.



4.5.5 ObjectType

The `ObjectType` provides a mapping between an OVAL Object, defined in content based on the OVAL Definitions Model, and the OVAL Items collected on the system under test.



Property	Type	Multiplicity	Description
id	oval:ObjectIDPattern	1	The globally unique identifier of an OVAL Object.
version	unsigned integer	1	The version of the globally unique OVAL Object.
variable_instance	unsigned integer	0..1	The unique identifier that differentiates between each unique instance of an OVAL Object. If an OVAL Object utilizes an OVAL Variable, a unique instance of each OVAL Object must be created for each OVAL Variable value. **Default Value:** *'1'*

comment	string	0..1	The documentation associated with the OVAL Object referenced by the id property.
flag	oval:FlagEnumeration	1	The outcome associated with OVAL Item collection.
message	oval:MessageType	0..*	Any messages that are relayed from a tool at run-time.
variable_value	VariableValueType	0..*	The value(s) associated with the variable(s) used by the OVAL Object referenced by the id property.
reference	ReferenceType	0..*	The identifiers of OVAL Items collected by the OVAL Object referenced by the id property.

4.5.6 VariableValueType

The `VariableValueType` identifies an *OVAL Variable* and value that is used by an OVAL Object during OVAL Item collection.

Property	Type	Multiplicity	Description
variable_id	oval:VariableIDPattern	1	The unique identifier of an OVAL Variable.
value	string	1	A value associated with the OVAL Variable identified by the `variable_id` property.

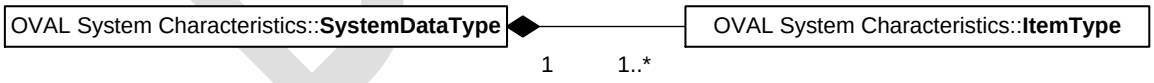
4.5.7 ReferenceType

The `ReferenceType` identifies an OVAL Item that was collected during OVAL Item collection.

Property	Type	Multiplicity	Description
item_ref	oval:ItemIDPattern	1	The unique identifier of an OVAL Item.

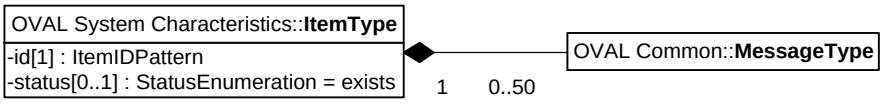
4.5.8 SystemDataType

The `SystemDataType` provides a container for all of the OVAL Items that were collected on the system under test.



4.5.9 ItemType

The `ItemType` is the abstract *OVAL Item* that defines the common properties associated with all OVAL Items defined in the OVAL System Characteristics OVAL Component Models.



Property	Type	Multiplicity	Description

id	oval:ItemIDPattern	1	The unique identifier of an OVAL Item. The id property is unique with in a given instantiation of the OVAL System Characteristics Model.
status	StatusEnumeration	0..1	The `status` property of an OVAL Item conveys the outcome of the system data collection effort. **Default Value:** *'exists'*
message	MessageType	0..50	Any messages that are relayed from a tool at run-time during the collection of an OVAL Item.

4.5.10 EntityAttributeGroup

The `EntityAttributeGroup` defines the properties that are common to all OVAL Item Entities in the OVAL Language.

Property	Type	Multiplicity	Description
datatype	oval:DatatypeEnumeration	0..1	The datatype for the entity. **Default Value:** *'string'*
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'*
status	StatusEnumeration	0..1	The status of the collection for an *OVAL Item* Entity. **Default Value:** *'exists'*

4.5.11 FlagEnumeration

The `FlagEnumeration` defines the acceptable outcomes associated with the collection of OVAL Items for a specified OVAL Object.

Enumeration Value	Description
error	This value indicates that an error prevented the determination of the existence of OVAL Items on the system.

complete	This value indicates that every matching OVAL Item on the system has been identified and represented in the OVAL System Characteristics. It can be assumed that no additional matching OVAL Items exist on the system.
incomplete	This value indicates that matching OVAL Items exist on the system, however, only a subset of those matching OVAL Items have been identified and represented in the OVAL System Characteristics. It cannot be assumed that no additional matching OVAL Items exist on the system.
does not exist	This value indicates that no matching OVAL Items were found on the system.
not collected	This value indicates that no attempt was made to collect OVAL Items on the system.
not applicable	This value indicates that the specified OVAL Object is not applicable to the system under test.

4.5.12 StatusEnumeration

The `StatusEnumeration` defines the acceptable status values associated with the collection of an OVAL Item or the properties of an OVAL Item.

Enumeration Value	Description
error	This value indicates that there was an error collecting an OVAL Item or a property of an OVAL Item.
exists	This value indicates that an OVAL Item, or a property of an OVAL Item, exists on the system and was collected.
does not exist	This value indicates that an OVAL Item, or a property of an OVAL Item, does not exist on the system.
not collected	This value indicates that no attempt was made to collect an OVAL Item or a property of an OVAL Item.

4.5.13 EntityItemSimpleBaseType

The `EntityItemSimpleBaseType` is an abstract type that defines a base type for all simple OVAL Item Entities.

Property	Type	Multiplicity	Description
attributes	EntityAttributeGroup	1	The standard attributes available to all entities.
value	string	0..1	The value of the entity. An empty string value SHOULD be used when a status other than 'exists' is specified.

4.5.14 EntityItemComplexBaseType

The `EntityComplexBaseType` is an abstract type that defines a base type for all complex OVAL Item Entities.

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

4.5.15 EntityItemIPAddressType

The `EntityItemIPAddressType` extends the `EntityItemSimpleBaseType` and describes an IPv4 or IPv6 IP address or prefix.

Property	Type	Multiplicity	Description
datatype	oval:SimpleDatatypeEnumeration	1	Possible values: • *'ipv4_address'* • *'ipv6_address'* Also allows an empty string value.

4.5.16 EntityItemIPAddressStringType

The `EntityItemIPAddressStringBaseType` extends the `EntityItemSimpleBaseType` and describes an IPv4 or IPv6 IP address or prefix or a string representation of the address.

Property	Type	Multiplicity	Description
datatype	oval:SimpleDatatypeEnumeration	1	Possible values: • *'ipv4_address'* • *'ipv6_address'* • *'string'* Also allows an empty string value.

4.5.17 EntityItemAnySimpleType

The `EntityItemAnySimpleType` extends the `EntityItemSimpleBaseType` and describes any simple data.

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

4.5.18 EntityItemBinaryType

The `EntityItemAnySimpleType` extends the `EntityItemSimpleBaseType` and describes any simple binary data.

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

4.5.19 EntityItemBoolType

The `EntityObjectBoolType` extends the `EntityItemSimpleBaseType` and describes any simple boolean data.

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

4.5.20 EntityItemFloatType

The `EntityItemFloatType` extends the `EntityItemSimpleBaseType` and describes any simple float data.

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

4.5.21 EntityItemIntType

The `EntityItemIntType` extends the `EntityItemSimpleBaseType` and describes any simple integer data.

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

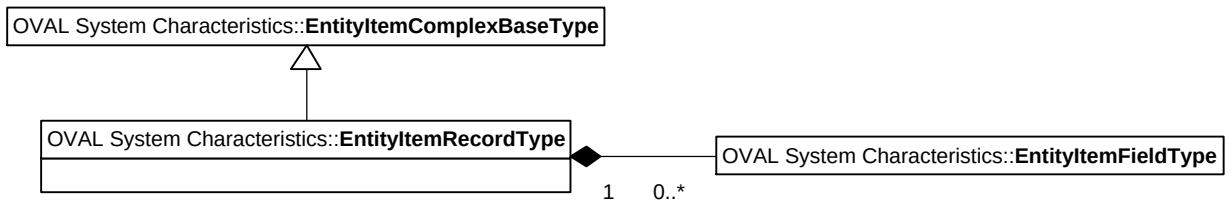
4.5.22 EntityItemStringType

The `EntityItemStringType` extends the `EntityItemSimpleBaseType` and describes any simple string data.

Property	Type	Multiplicity	Description
datatype	oval:SimpleDatatypeEnumeration	0..1	This value is fixed as *'string'*.

4.5.23 EntityItemRecordType

The `EntityItemRecordType` extends the `EntityItemComplexBaseType` 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	Multiplicity	Description
datatype	oval:ComplexDatatypeEnumeration	0..1	This value is fixed as *'record'*.
field	EntityStateFieldType	0..*	Defines the name of the field whose value will be assessed.

4.5.24 EntityItemFieldType

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

Property	Type	Multiplicity	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.
value	string	0..1	The value of the field. An empty string value SHOULD be used when a status other than 'exists' is specified.

4.5.25 EntityItemVersionType

The `EntityItemVersionType` extends the `EntityItemSimpleBaseType` and describes a version string data.

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

4.5.26 EntityItemFileSetRevisionType

The `EntityItemFileSetRevisionType` extends the `EntityItemSimpleBaseType` and describes a file set revision string data.

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

4.5.27 EntityItemIOSVersionType

The `EntityItemIOSVersionType` extends the `EntityItemSimpleBaseType` and describes a Cisco IOS version string data.

Property	Type	Multiplicity	Description
datatype	oval:SimpleDatatypeEnumeration	1	Possible values: • *'ios_version'* • *'string'* The string type is an option in order to allow use of regular expressions.

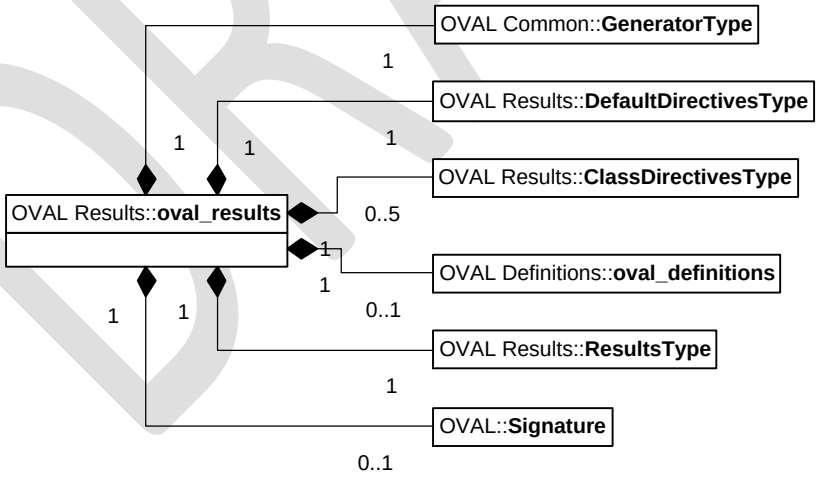
4.5.28 EntityItemEVRStringType

The `EntityItemEVRStringType` extends the `EntityItemSimpleBaseType` and describes an EPOCH:VERSION-RELEASE string data.

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

4.6 OVAL Results Model

The OVAL Results Model is used to report the results of an evaluation of a set of systems based upon a set of OVAL Definitions leveraging the OVAL System Characteristics. In this way, the OVAL Results Model provides detailed information about the set of assertions that were evaluated, the observed states of the evaluated systems, and the detailed results of the evaluation.

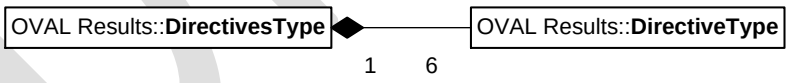


Property	Type	Multiplicity	Description
generator	oval:GeneratorType	1	Information regarding the generation of the OVAL Results content. The `timestamp` property of the `generator` MUST represent the time at

			which the `oval_results` was created.
directives	DefaultDirectivesType	1	Describes the default set of directives that specify the results that have been included in the OVAL Results. The `default_directives` MUST be used for any OVAL Definitions `result` value that is not overridden by a `class_directives` construct.
class_directives	ClassDirectivesType	0..5	Describes the set of directives that specify the class-specific results that have been included in the OVAL Results. The `class_directives` MAY be used to override the default directives.
oval_definitions	oval-def:oval_definitions	0..1	The source OVAL Definitions used to generate the OVAL Results.
results	ResultsType	1	Contains the evaluation results for all OVAL Definitions on all systems under test.
signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the OVAL Results content.

4.6.1 DirectivesType

The `DirectivesType` defines what result information has been included, and to what level of detail, in the OVAL Results, for each possible result value defined in the `ResultEnumeration`.

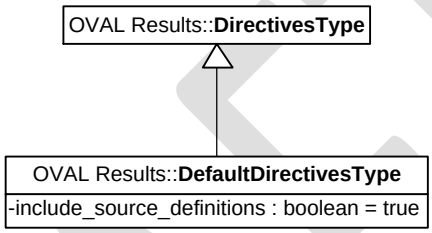


Property	Type	Multiplicity	Description
definition_true	DirectiveType	1	Defines what result information has been included for OVAL Definitions that evaluate to *'true'*.
definition_false	DirectiveType	1	Defines what result information has been included for OVAL Definitions that evaluate to *'false'*.
definition_unknown	DirectiveType	1	Defines what result information has been included for OVAL Definitions that evaluate to *'unknown'*.
definition_error	DirectiveType	1	Defines what result information has been

			included for OVAL Definitions that evaluate to *'error'*.
definition_not_evaluated	DirectiveType	1	Defines what result information has been included for OVAL Definitions that evaluate to *'not evaluated'*.
definition_not_applicable	DirectiveType	1	Defines what result information has been included for OVAL Definitions that evaluate to *'not applicable'*.

4.6.2 DefaultDirectivesType

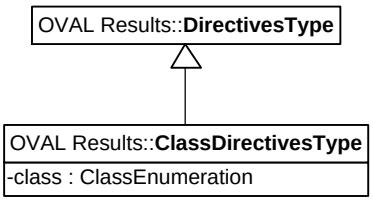
The `DefaultDirectivesType` defines the result information to include in the OVAL Results for all *OVAL Definitions* regardless of class as defined in the `ClassEnumeration`.



Property	Type	Multiplicity	Description
include_source_definitions	boolean	0..1	Specifies whether or not the source OVAL Definitions are included in the OVAL Results. When *'true'* the source OVAL Definitions MUST be included in the OVAL Results. When *'false'* the source OVAL Definitions MUST NOT be included in the OVAL Results. **Default Value:** *'true'*

4.6.3 ClassDirectivesType

The `ClassDirectivesType` defines the result information to include in the OVAL Results for a specific class of OVAL Definitions as defined in the `ClassEnumeration`. Please note that this will override the directives in the `DefaultDirectivesType` for the specified class.



Property	Type	Multiplicity	Description
class	oval:ClassEnumeration	1	Specifies the class of OVAL Definitions to which the

			defined OVAL Results directives will be applied.

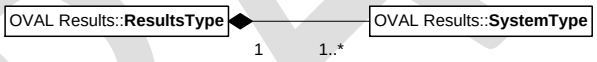
4.6.4 DirectiveType

The `DirectiveType` defines what result information, and to what level of detail, is included in OVAL Results.

Property	Type	Multiplicity	Description
reported	boolean	1	Specifies whether or not OVAL Definitions, with the specified result, should be included in the OVAL Results. If the `reported` property is set to *'true'*, OVAL Definitions that evaluate to the specified result MUST be included in the OVAL Results. If the `reported` property is set to *'false'*, OVAL Definitions that evaluate to the specified result MUST NOT be included in the OVAL Results.
content	ContentEnumeration	0..1	Specifies the level of detail that is included in the OVAL Results. **Default Value:** *'full'*

4.6.5 ResultsType

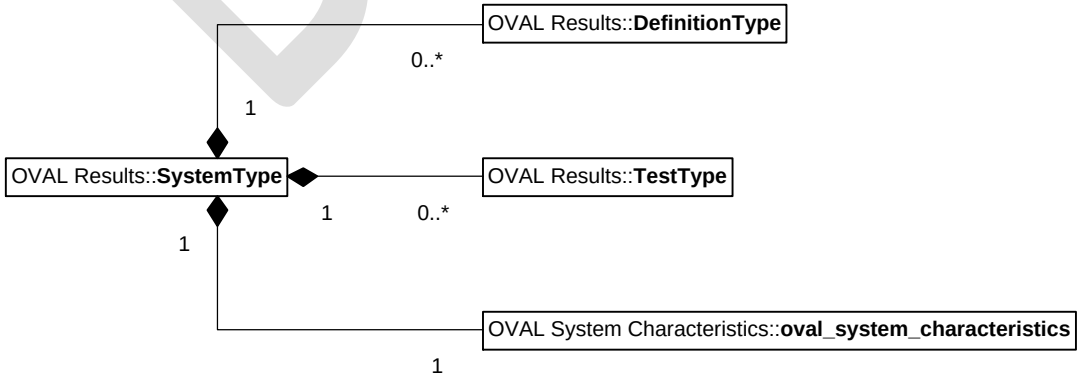
The `ResultsType` contains the evaluation results for all OVAL Definitions on all systems under test.



Property	Type	Multiplicity	Description
results	SystemType	1..*	The evaluation results for all OVAL Definitions on each system under test.

4.6.6 SystemType

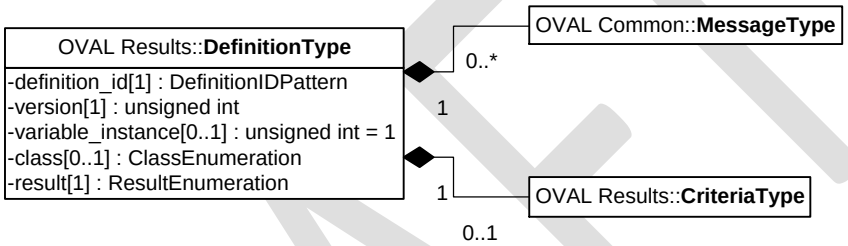
The `SystemType` provides the evaluation results for the OVAL Definitions and OVAL Tests as well the OVAL System Characteristics for an individual system.



Property	Type	Multiplicity	Description
definitions	DefinitionType	0..*	The evaluation results of the OVAL Definitions.
tests	TestType	0..*	The evaluation results of the OVAL Tests.
system_characteristics	oval-sc:oval_system_characteristics	1	A copy of the OVAL System Characteristics that were evaluated against the OVAL Definitions to produce the OVAL Results.

4.6.7 DefinitionType

The `DefinitionType` contains the results of the evaluation of an OVAL Definition.

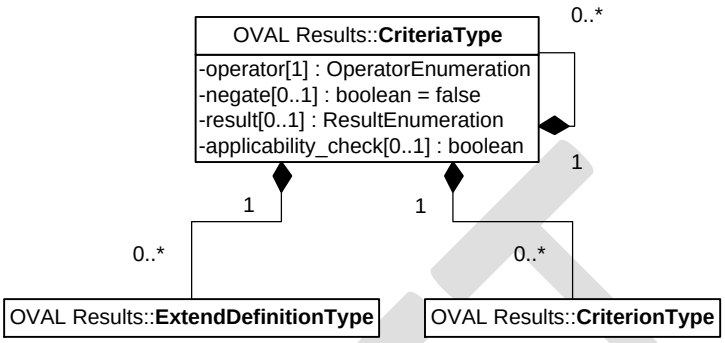


Property	Type	Multiplicity	Description
definition_id	oval:DefinitionIDPattern	1	The unique identifier of an OVAL Definition that was used to generate the OVAL Results.
version	unsigned int	1	The version of the globally unique OVAL Definition.
variable_instance	unsigned int	0..1	The unique identifier that differentiates between each unique instance of an OVAL Definition. If an OVAL Definition utilizes an OVAL Variable, a unique instance of each OVAL Definition must be created for each collection of values assigned to the OVAL Variable. **Default Value:** *'1'*
class	oval:ClassEnumeration	0..1	The class of the OVAL Definition.
result	ResultEnumeration	1	The result of the evaluation of the OVAL Definition.
message	oval:MessageType	0..*	Any messages that are relayed from a tool at run-time during the evaluation of an OVAL Definition.
criteria	CriteriaType	0..1	Contains the individual results of the logical statements that form the

			OVAL Definition.

4.6.8 CriteriaType

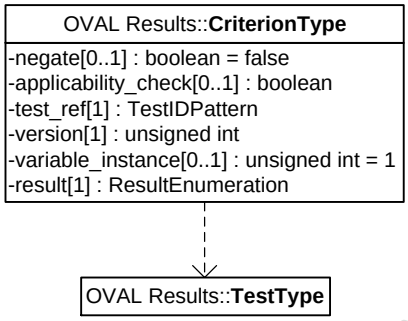
The `CriteriaType` combines the logical statements that form the OVAL Definition.



Property	Type	Multiplicity	Description
operator	oval:OperatorEnumeration	1	The logical operator that is used to combine the individual results of the logical statements defined by the `child_criteria` property.
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'*
result	ResultEnumeration	1	The evaluation result after the `operator` property and `negate` property have been applied.
criteria	CriteriaType	1..*	Logical statements that will be combined according to the `operator` property.
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'*.

4.6.9 CriterionType

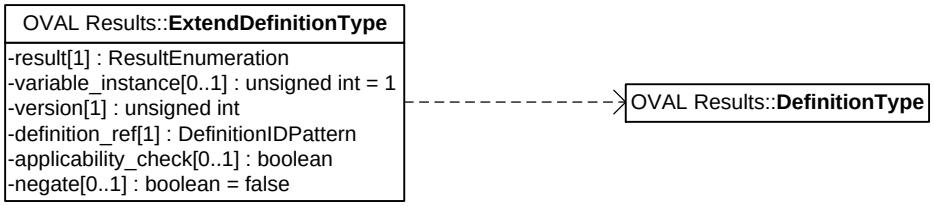
The `CriterionType` is a logical statement that references an OVAL Test from an OVAL Definition.



Property	Type	Multiplicity	Description
test_ref	oval:TestIDPattern	1	The unique identifier of an OVAL Test contained in the OVAL Definitions used to generate the OVAL Results.
version	unsigned int	1	The version of the globally unique OVAL Test referenced by the `test_ref` property.
variable_instance	unsigned int	0..1	The unique identifier that differentiates between each unique instance of an OVAL Test. If an OVAL Test utilizes an OVAL Variable, a unique instance of each OVAL Test must be created for each collection of values assigned to the OVAL Variable. **Default Value:** *'1'*
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'*
result	ResultEnumeration	1	The evaluation result of the OVAL Test, referenced by the `test_ref` property, after the `negate` property has been applied.
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'*.

4.6.10 ExtendDefinitionType

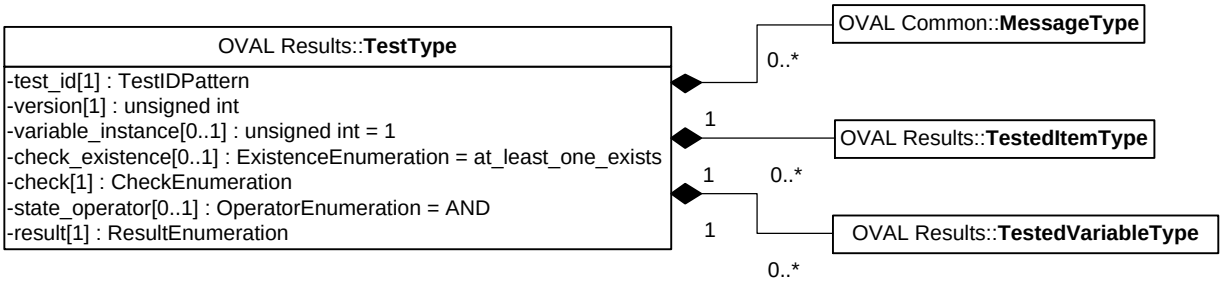
The `ExtendDefinitionType` is a logical statement that references another OVAL Definition.



Property	Type	Multiplicity	Description
definition_ref	oval:DefinitionIDPattern	1	The unique identifier of an OVAL Definition used to generate the OVAL Results.
version	unsigned int	1	The version of the globally unique OVAL Definition referenced by the `definition_ref` property.
variable_instance	unsigned int	0..1	The unique identifier that differentiates between each unique instance of an OVAL Definition. If an OVAL Definition utilizes an OVAL Variable, a unique instance of each OVAL Definition must be created for each collection of values assigned to the OVAL Variable. **Default Value:** *'1'*
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'*
result	ResultEnumeration	1	The evaluation result of the OVAL Definition, referenced by the `definition_ref` property, after the `negate` property has been applied.
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'*.

4.6.11 TestType

The `TestType` contains the result of an *OVAL Test*.

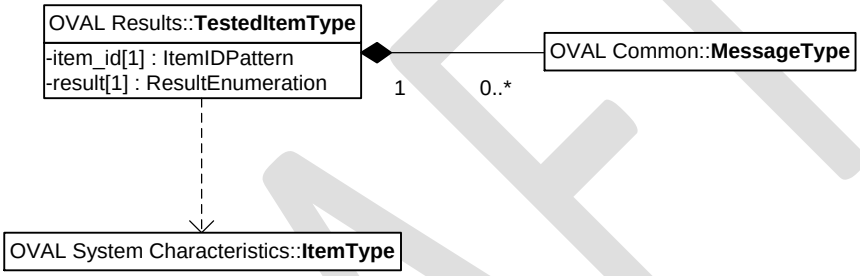


Property	Type	Multiplicity	Description
test_id	oval:TestIDPattern	1	The unique identifier of an OVAL Test contained in the OVAL Definitions used to generate the OVAL Results.
version	unsigned int	1	The version of the globally unique OVAL Test referenced by the `test_id` property.
variable_instance	unsigned int	0..1	The unique identifier that differentiates between each unique instance of an OVAL Test. If an OVAL Test utilizes an OVAL Variable, a unique instance of each OVAL Test must be created for each collection of values assigned to the OVAL Variable. **Default Value:** *'1'*
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'*
result	ResultEnumeration	1	The evaluation result of the OVAL Test referenced by the `test_id` property.

message	oval:MessageType	0..*	Any messages that are relayed from a tool at run-time during the evaluation of an OVAL Test.
tested_item	TestedItemType	0..*	Specifies a reference to each OVAL Item used in the evaluation of an OVAL Test.
tested_variable	TestedVariableType	0..*	Specifies each OVAL Variable value used in the evaluation of an OVAL Test. This includes the OVAL Variable values used in both OVAL Objects and OVAL States.

4.6.12 TestedItemType

The `TestedItemType` contains the result of evaluating a collected OVAL Item against the OVAL State(s), if any, as specified by the corresponding OVAL Test.



Property	Type	Multiplicity	Description
item_id	oval:ItemIDPattern	1	The unique identifier of an OVAL Item collected during OVAL Item Collection.
result	ResultEnumeration	1	The evaluation result of the OVAL Item against the OVAL State(s), if any, as specified by the corresponding OVAL Test.
message	oval:MessageType	0..*	Any messages that are relayed from a tool at run-time during the evaluation of an OVAL Item against an OVAL State.

4.6.13 TestedVariableType

The `TestedVariableType` specifies the value of an OVAL Variable used during the evaluation of an OVAL Test.

Property	Type	Multiplicity	Description
variable_id	oval:VariableIDPattern	1	The unique identifier of an OVAL Variable.
value	Any	1	A value of the OVAL Variable referenced by the `variable_id` property.

4.6.14 ContentEnumeration

The `ContentEnumeration` defines the acceptable levels of detail for the result information included in the OVAL Results.

Enumeration Value	Description
thin	This value indicates that only the minimal amount of information is represented in the OVAL Results. Specifically: • The `definition_id` property of `DefinitionType` will be included. • The `result` property of `DefinitionType` will be included. • The `criteria` property of `DefinitionType` will not be included. • The `collected_objects` and `system_data` properties, of the `system_characteristics` property in `SystemType`, will not be included.
full	This value indicates that a full detailed result of information is represented in the OVAL Results. Specifically: • The `definition_id` property of `DefinitionType` will be included. • The `result` property of `DefinitionType` will be included. • The `criteria` property of `DefinitionType` will be included. • The `collected_objects` and `system_data` properties, of the `system_characteristics` property in `SystemType`, will be included. The value *'full'* is equivalent to *'thin'* with the `collected_objects` and `system_data` properties, of the `system_characteristics` property in `SystemType`, included.

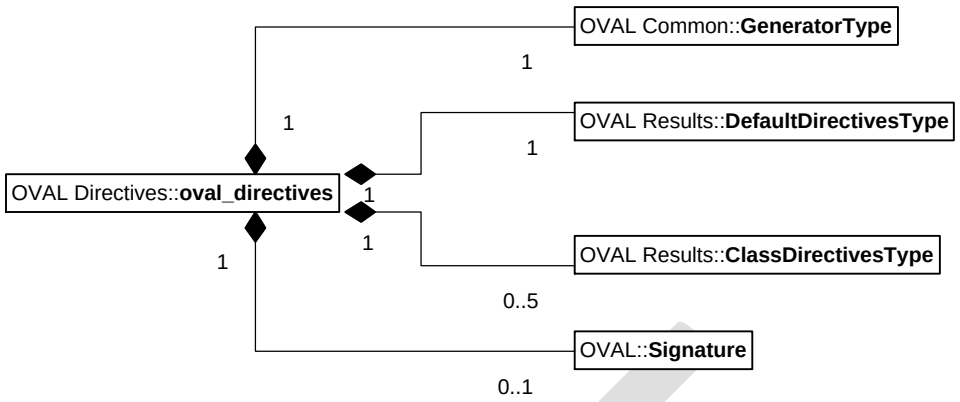
4.6.15 ResultEnumeration

The `ResultEnumeration` defines the acceptable evaluation result values in the OVAL Language.

Enumeration Value	Description
true	This value indicates that the conditions of the evaluation were satisfied.
false	This value indicates that the conditions of the evaluation were not satisfied.
unknown	This value indicates that it could not be determined if the conditions of the evaluation were satisfied.
error	This value indicates that an error occurred during the evaluation.
not evaluated	This value indicates that a choice was made not to perform the evaluation.
not applicable	This value indicates that the evaluation being performed does not apply to the given platform.

4.7 OVAL Directives Model

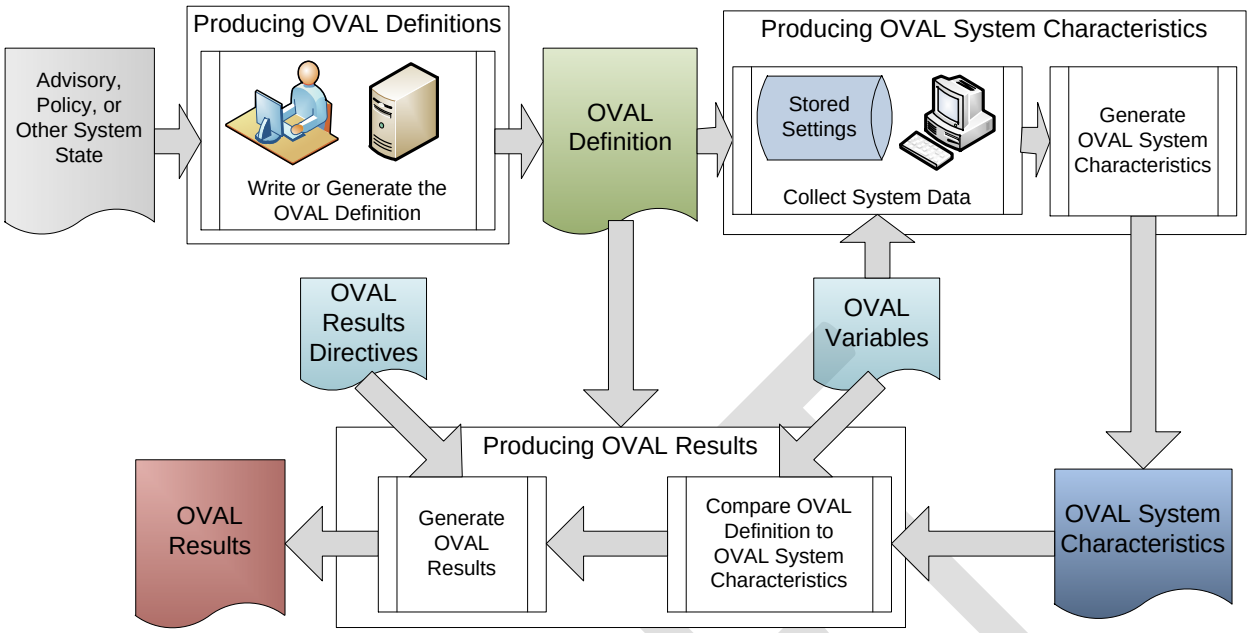
The OVAL Directives Model is used to control what result information is included in the OVAL Results as well as specify its level of detail.



Property	Type	Multiplicity	Description
generator	oval:GeneratorType	1	Information regarding the generation of the OVAL Directives content. The `timestamp` property of the `generator` MUST represent the time at which the `oval_directives` was created.
directives	oval-res:DefaultDirectivesType	1	Describes the default set of directives that specify the results that have been included in the OVAL Results.
class_directives	oval-res:ClassDirectivesType	0..5	Describes the set of directives that specify the class-specific results that have been included in the OVAL Results.
signature	ext:Signature	0..1	Mechanism to ensure the integrity and authenticity of the OVAL Directives content.

5 Processing Model for the OVAL Language

The processing section describes in detail how the major components of the OVAL Language Data Model are used to produce OVAL Definitions, OVAL System Characteristics, and OVAL Results. The diagram below provides an overview of the complete process and highlights the major activities of this process.



When producing OVAL Definitions there is assumed to be an advisory, policy, or other system state description that is either manually translated or automatically processed to create an OVAL Definition. The resulting OVAL Definition is based upon the low level system information that was specified and will be the basis for both producing OVAL System Characteristics and producing OVAL Results.

OVAL System Characteristics are produced by collecting system data directly from an end system on some other configuration information data store. This data collection subprocess can be controlled by the OVAL Objects specified in the OVAL Definition, or by any other selection method. When OVAL Objects are used to guide the data collection process OVAL Variables may be supplied to allow for tailoring of the OVAL Objects. The collected system data is then compiled into OVAL System Characteristics that includes information about the tool that collected the data and the set of OVAL Objects that were used to guide data collection, if any.

OVAL Results are produced by comparing an OVAL Definition and the system state that it describes to some observed system state as represented in OVAL System Characteristics. This comparison process, referred to as Definition Evaluation, can be tailored by OVAL Variables and creates detailed assessment results which are then used to generate OVAL Results. OVAL Results include information about the tool that produced them and varying levels of assessment result information as specified by a set of OVAL Directives.

5.1 Producing OVAL Definitions

Producing OVAL Definitions is the process by which information from some source external to OVAL is consumed by a person, tool, or service and then transformed into an OVAL Definition. Often this information comes from a security advisory, configuration checklist, or other data feed. Other times this information must be created through detailed system investigation and research of known issues. In

either case, low level system state information is encoded in the form of an assertion about a system state.

5.1.1 Reuse of Definition, Test, Object, State, and Variable

The OVAL Language enables content reuse through the use of globally unique IDs. When producing OVAL Definitions, OVAL Tests, OVAL Objects, OVAL States, and OVAL Variables, existing content SHOULD be reused when possible.

5.1.2 Tracking Change

The version property provides the ability to track changes to OVAL Definitions, OVAL Tests, OVAL Objects, OVAL States, and OVAL Variables. Proper usage of the version property is critical for content sharing and reuse. When updating an OVAL Definition, OVAL Test, OVAL Object, OVAL State, or OVAL Variable the version property MUST be incremented for each revision.

5.1.3 Metadata

Each OVAL Definition, as defined by the `oval-def:DefinitionType`, includes a `metadata` property. The contents of the `metadata` property MUST NOT impact OVAL Definition evaluation. All information that is encoded in the `metadata` property SHOULD also be encoded in the OVAL Definition's `criteria`.

5.1.3.1 Authoritative References

The `reference` property of an OVAL Definition's `metadata` property SHOULD provide an authoritative citation for the specific system state being described by the OVAL Definition. OVAL Definitions with a `class` property value of *'vulnerability'* SHOULD include a reference to the CVE Name for the vulnerability when one exists. OVAL Definitions with a `class` property value of *'compliance'* SHOULD include a reference to the CCE Name for the configuration item when one exists. OVAL Definitions with a `class` property value of *'inventory'* SHOULD include a reference to the CPE for the relevant operating system or application when a CPE Name exists.

5.1.3.2 Platforms and Products

The `platform` and `product` properties of an OVAL Definition's `metadata` property SHOULD provide a listing of platforms and products to which the OVAL Definition is known to apply.

5.1.4 Content Integrity and Authenticity

Content expressed in the OVAL Definitions Model MAY be digitally signed in order to preserve content integrity and authenticity. The OVAL Definitions Model defines six locations for including a digital signature. Any of these locations MAY be used. See section 6.1 XML Signature Support.

5.2 Producing OVAL System Characteristics

Producing OVAL System Characteristics is the process by which detailed system state information is collected and represented in a standard format. This information may be collected through direct interaction with an end system by using system APIs to query the state of the system, or by gathering

information from some other source of system state information, like a configuration management database.

5.2.1 System Information

The `oval-sc:system_info` property of the OVAL System Characteristics model MUST accurately represent the system from which the data was collected. When the system data was collected from a source other than directly from the system being described, the `oval-sc:system_info` type MUST represent the original system from which the data was collected.

5.2.2 Collected Objects

When a set of OVAL Objects is used to guide the collection of system data, the OVAL Objects that were used MUST be recorded as `objects` in the `oval-sc:collected_objects` property of the OVAL System Characteristics model. This section describes the process of creating an `oval-sc:object` in the collection of `oval-sc:collected_objects`.

5.2.2.1 flag Usage

Each `object` listed in the `oval-sc:collected_objects` MUST specify the outcome of the data collection effort by setting the flag `property` to the appropriate value. The valid flag values are defined in the `oval-sc:FlagEnumeration`. The correct usage of the flag enumeration values in the context of the `flag` property is specified in the following table.

Enumeration Value	When to Use the Enumeration Value?
error	This value MUST be used when an error that prevents the collection of the OVAL Items for the OVAL Object. The `object` property SHOULD include one or more `messages` describing the error condition.
complete	This value MUST be used when the collection process for the OVAL Object was successful and accurately captured the complete set of matching OVAL Items.
incomplete	This value MUST be used when the collection process for the OVAL Object was successful but the complete set of matching OVAL Items is not represented by the set of `references`. The `object` property SHOULD include one or more `messages` explaining the incomplete `flag` value.
does not exist	This value MUST be used when no matching OVAL Items were found.
not collected	This value MUST be used when no attempt was made to collect the OVAL Object. The `object` property MAY include one or more `messages` explaining the not collected `flag` value.
not applicable	This value MUST be used the specified OVAL Object is not applicable to the system under test.

	The `object` property MAY include one or more `messages` explaining the not applicable `flag` value.

5.2.2.2 variable_instance property

When an OVAL Object makes use of an OVAL Variable, either directly or indirectly, each collection of values assigned to the OVAL Variable MUST be differentiated by incrementing the `variable_instance` property once for each assigned collection of values for the OVAL Variable. When more than one collection of values is assigned to an OVAL Variable, a given OVAL Object will appear as a `oval-sc:collected_object` once for each assigned value.

5.2.2.3 Item References

Each OVAL Item that is collected as a result of collecting a given OVAL Object MUST be referenced by the `reference` property of the `object`. A given OVAL Item MAY be referenced by one or more `objects`. This situation will occur when two distinct OVAL Objects identify overlapping sets of OVAL Items.

When the `flag` property has a value of *'not collected'* or *'not applicable'* the `object` MUST NOT include any OVAL Item references.

5.2.2.4 Variable Values

Each OVAL Variable and its value used when collecting OVAL Items for an OVAL Object MUST be recorded in the `variable_value` property of the `object`.

5.2.3 Conveying System Data without OVAL Objects

OVAL Objects are commonly used to guide the collection of OVAL Items. However, system state information may be collected without the use of OVAL Objects. OVAL Items MAY be collected by searching system data stores, API calls, algorithms, or other proprietary processes. When this is done, the OVAL System Characteristics will not contain a `collected_objects` section, however, it will contain a `system_data` section with all of the OVAL Items collected.

5.2.4 Recording System Data and OVAL Items

The `system_data` property holds a collection of OVAL Items. This section describes the process of building an OVAL Item and the constraints that apply to OVAL Items.

5.2.4.1 Item IDs

Each OVAL Item contains a unique identifier which distinguishes it from other OVAL Items that are represented in the collection of `system_data`. Item IDs MUST be unique within an OVAL System Characteristics Model.

5.2.4.2 Unique Items

OVAL Items are differentiated by examining each OVAL Item's name and each of the OVAL Item's entity names and values. Each OVAL Item MUST represent a unique system data artifact. No two OVAL Items within an OVAL System Characteristics Model can be the same.

5.2.4.3 Partial Matches

A partial match is when an OVAL Item, containing some information, is reported in the OVAL System Characteristics rather than simply not reporting the OVAL Item. Partial matches are useful for debugging purposes when an OVAL Item does not exist on the system or is not collected due to limitations in the OVAL Capable Product. Please note that the use of partial matches is optional.

5.2.4.4 Item Status

The valid status values, for an OVAL Item, are defined in the `oval-sc:StatusEnumeration`. The correct usage of the status enumeration values in the context of the `status` property is specified in the following table.

Enumeration Value	When to Use the Enumeration Value?
error	This value MUST be used when there is an error that prevents the collection of an OVAL Item or any of its entities. The OVAL Item SHOULD include one or more `messages` describing the error condition.
exists	This value MUST be used when an OVAL Item is successfully collected.
does not exist	This value MUST be used when the OVAL Item is not found on the system being examined. The use of this value is optional and is only used to report a partial match. If a partial match is not being reported, the OVAL Item MUST NOT be reported in the OVAL System Characteristics. The OVAL Item MAY include one or more `messages` describing this status value.
not collected	This value MUST be used when no attempt is made collect the OVAL Item. The use of this value is optional and is only used to report a partial match. If a partial match is not being reported, the OVAL Item MUST NOT be reported in the OVAL System Characteristics. The OVAL Item SHOULD include one or more `messages` describing this status value.

5.2.4.5 Item Entities

OVAL Item Entities must be added to the OVAL Item such that it aligns with the constraints specified in the appropriate OVAL Component Model and the requirements in this section.

5.2.4.5.1 Determining Which Entities to Include

OVAL Component Models define concrete OVAL Items and their entities. All entities within an OVAL Item are optional. When creating an OVAL Item any number of item entities MAY be included. However, sufficient OVAL Item entities MUST be included to ensure that the OVAL Item describes only a single system configuration item.

Many OVAL Items include entities that have dependencies upon other entities within the same OVAL Item. When dependencies exist between OVAL Item entities, if an entity is included then all entities that it depends upon MUST also be included in the OVAL Item. When using OVAL Objects to guide the collection of system data, the entities included in the OVAL Object SHOULD be included in the OVAL Items that it identifies.

When collecting system data an OVAL State MAY be used to determine which entities to include within and OVAL Item. This sort of processing can be an optimization made when collecting data. For example, if the OVAL State makes an assertion about a single entity it may not be necessary to include all other OVAL Item entities.

5.2.4.5.2 Status

The OVAL Item Entity `status` conveys the outcome of attempting to collect one property of a piece of system state information. The valid OVAL Item Entity status values are defined in the `oval-sc:StatusEnumeration`. The status of an OVAL Item Entity can be independent of other OVAL Item Entities and SHOULD NOT be propagated up to the containing OVAL Item. The following table indicates when to use each status value.

Enumeration Value	When to Use the Enumeration Value?
error	This value MUST be used when there is an error that prevents the collection of the OVAL Item Entity.
exists	This value MUST be used when the OVAL Item Entity exists on the system and is collected.
does not exist	This value MUST be used when the OVAL Item Entity does not exist on the system.
not collected	This value MUST be used when no attempt is made to collect the OVAL Item Entity.

5.2.4.5.3 Datatype

The `datatype` of the OVAL Item Entity describes how the `value` of the OVAL Item Entity should be interpreted. The valid `datatype` values for an OVAL Item Entity are listed in the `oval:DatatypeEnumeration` and restricted as needed in OVAL Component Models. When assigning a `datatype` to an OVAL Item Entity, there are two cases to consider:

1. The `datatype` is fixed to a specific `datatype` value. In this case, the OVAL Item Entity MUST always use the specified `datatype` value.
2. The `datatype` can be one of several datatype values. In this case, the `datatype` value that most appropriately describes the value of the OVAL Item Entity SHOULD be used. If an OVAL Item Entity value is not present, the `datatype` value must be set to the default datatype value specified in corresponding OVAL Component Model.

5.2.4.5.4 Value

The final aspect of an OVAL Item Entity is its `value`. An OVAL Item Entity may contain simple character data or complex structured data as specified in the corresponding OVAL Component Model. All OVAL Item Entity values must conform to the constraints defined in the `oval-sc:DatatypeEnumeration`.

5.2.5 Content Integrity and Authenticity

Content expressed in the OVAL System Characteristics Model MAY be digitally signed in order to preserve content integrity and authenticity. See Section 6.1 on XML Signature Support.

5.3 Producing OVAL Results

Producing OVAL Results is the process by which detailed system state information is evaluated against the expected state of a system and represented in a standardized format. This standardized format conveys the results of the evaluation which can indicate the presence of a vulnerability, compliance to a policy, installation of software, or even the presence of malware artifacts. Additionally, the results can be consumed by other tools where they can be interpreted and used to inform remediation of discovered issues.

5.3.1 Definition Evaluation

OVAL Definition Evaluation is the process examining the characteristics of a system and applying one or more logical statements about those characteristics to determine an overall result for the system state that the OVAL Definition describes. Each OVAL Definition has zero or one logical `criteria` components, which are combined using logical operators, such as '`AND`' and '`OR`'. The overall result of evaluating an OVAL Definition is determined by evaluating its `criteria` component. This process is described in detail in the following section.

5.3.1.1 Evaluating a Deprecated OVAL Definition

When evaluating a deprecated OVAL Definition, that does not have a `criteria` construct, the OVAL Definition MUST evaluate to *'not evaluated'*. If a deprecated OVAL Definition contains a `criteria` construct, the OVAL Definition SHOULD evaluate as if it were not deprecated. However, the OVAL Definition MAY evaluate to *'not evaluted'*.

5.3.1.2 Criteria Evaluation

A `criteria` component of an OVAL Definition combines one or more logical statements in order to determine a result value. A `criteria` can be made up of other `criteria`, `criterion`, or `extend_definitions`, along with an `operator` property that specifies how to logically combine the specified logical statements. For more information on how to combine the individual results of the logical statements specified within a `criteria`, see Section 5.3.6.2. The result value of the criteria is determined by first evaluating the `operator` property to combine the logical statements and then evaluating the `negate` property. See Section 5.3.1.5 for additional information on how to negate the result of the `criteria`.

5.3.1.2.1 applicability_check

If a value for the `applicability_check` property is specified on the `criteria` construct, in an OVAL Definition, the `applicability_check` property and value MUST be replicated on the `criteria` construct in the OVAL Results.

5.3.1.3 Criterion Evaluation

The result of a `criterion` construct is the result of the OVAL Test that it references, after the `negate` property has been applied. See Section 5.3.1.5 Negate Evaluation for additional information on how to negate the result of an OVAL Test.

The `variable_instance` property of the `criterion` is carried over from the `variable_instance` value of the referenced OVAL Test.

5.3.1.3.1 applicability_check

If a value for the `applicability_check` property is specified on the `criterion` construct, in an OVAL Definition, the `applicability_check` property and value MUST be replicated on the `criterion` construct in the OVAL Results.

5.3.1.4 Extend Definition Evaluation

The result of an `extend_definition` construct is the result of the OVAL Definition, that it references, after the `negate` property has been applied. See Section 5.3.1.5 Negate Evaluation for additional information on how to negate the result of an OVAL Definition.

The `variable_instance` property of the `extend_definition` is carried over from the `variable_instance` value of the referenced OVAL Definition.

5.3.1.4.1 applicability_check

If a value for the `applicability_check` property is specified on the `extend_definition` construct, in an OVAL Definition, the `applicability_check` property and value MUST be replicated on the `extend_definition` construct in the OVAL Results.

5.3.1.5 Negate Evaluation

When the `negate` property is *'true'*, the final result of a construct MUST be the logical complement of its result value. That is, for any construct that evaluates to *'true'*, the final result would become *'false'*, and vice versa. The `negate` property does not apply to non-Boolean result values. If a non-Boolean result value is encountered, the final result MUST be the non-Boolean result value. If the negate property is set to *'false'*, the final result of a construct will be its original result value.

5.3.1.6 Variable Instance

The value of the `variable_instance` property is derived from the `variable_instance` values of the OVAL Definitions and OVAL Tests that are referenced within the OVAL Definition's `criteria`. When an OVAL Definition references another OVAL Definition or an OVAL Test that makes use of an OVAL Variable, each collection of values assigned to the OVAL Variable MUST be differentiated by

incrementing the `variable_instance` property. The `variable_instance` value is incremented once for each assigned collection of values for the OVAL Variable. When more than one collection of values is assigned to an OVAL Variable, an OVAL Definition will appear in the `definitions` section once for each assigned collection of values.

5.3.2 Test Evaluation

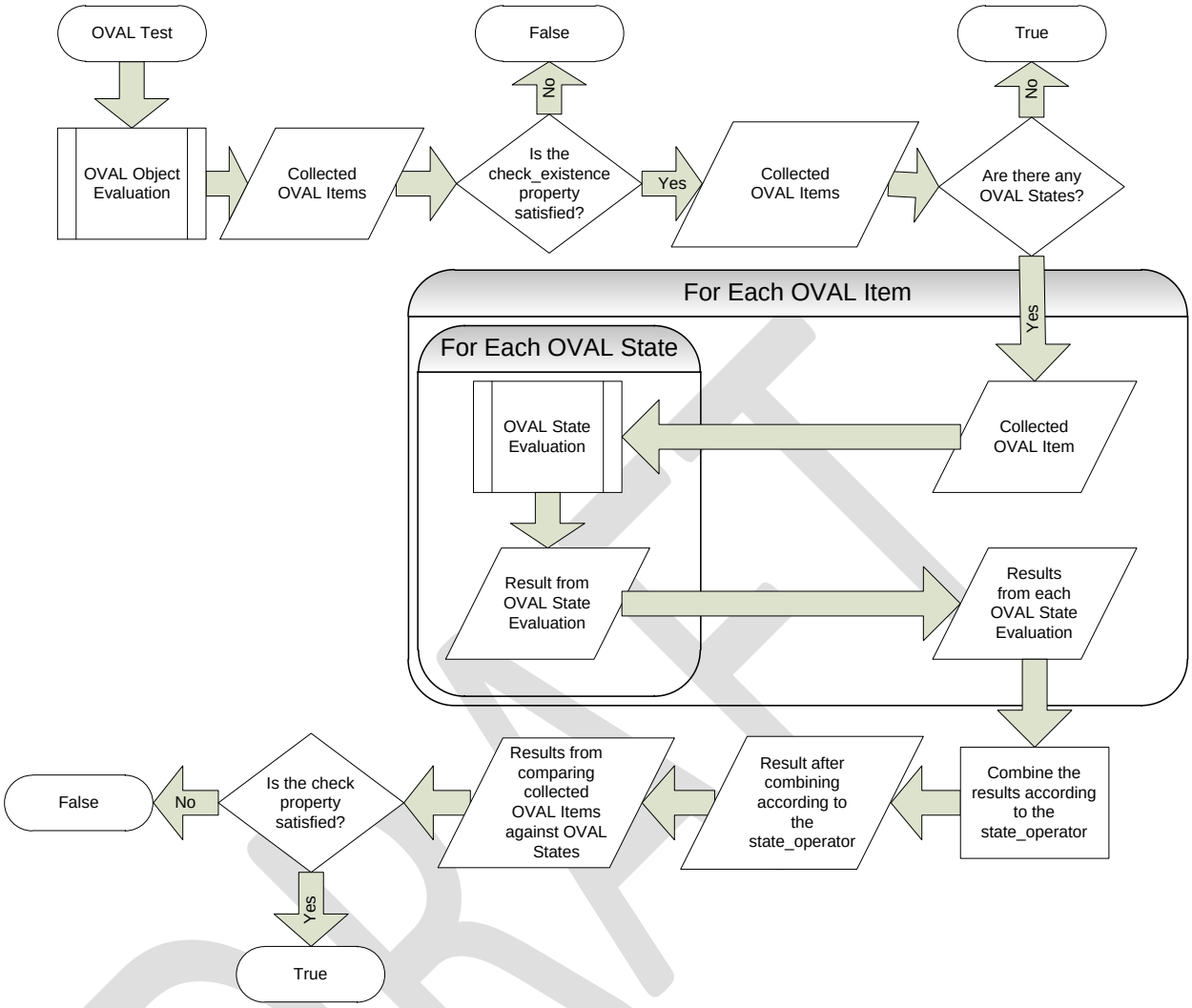
An OVAL Test is the standardized representation of an assertion about the state of a system. An OVAL Test contains references to an OVAL Object that specifies which system data to collect and zero or more OVAL States that specify the expected state of the collected system data. OVAL Test Evaluation is the process of comparing the collected set of system data, as OVAL Items, to zero or more OVAL States.

The result of the OVAL Test Evaluation is then determined by combining the results of the following three test evaluation parameters:

1. **Existence Check Evaluation** – The process of determining whether or not the number of OVAL Items, that match the specified OVAL Object, satisfy the requirements specified by the `check_existence` property.

2. **Check Evaluation** – The process of determining whether or not the number of collected OVAL Items, specified by the check property, match the specified OVAL States.

3. **State Operator Evaluation** – The process of combining the individual results, from the comparison of an OVAL Item to the specified OVAL States, according to the `state_operator` property.



5.3.2.1 *Existence Check Evaluation*

Existence Check Evaluation is the process of determining whether or not the number of OVAL Items, that match the specified OVAL Object, satisfy the requirements specified by the `check_existence` property. The `check_existence` property specifies how many OVAL Items that match the specified OVAL Object must exist on the system in order for the OVAL Test to evaluate to *'true'*. To determine if the `check_existence` property is satisfied, the status of each OVAL Item collected by the OVAL Object must be examined.

The following tables describe how each `ExistenceEnumeration` value affects the result of the Existence Check Evaluation. The far left column identifies the `ExistenceEnumeration` value in question, and the middle column specifies the different combinations of individual OVAL Item status values that may be found. The last column specifies the final result of the Existence Check Evaluation according to the combination of individual OVAL Item status values.

Enumeration Value	Number of Individual Item Status Values				Existence Result
all_exist	exists	does not exist	error	not collected	

	1+	0	0	0	true
	0	0	0	0	false
	0+	1+	0+	0+	false
	0+	0	1+	0+	error
	0+	0	0	1+	unknown
	--	--	--	--	not evaluated
	--	--	--	--	not applicable

Enumeration Value	Number of Individual Item Status Values				Existence Result
any_exist	exists	does not exist	error	not collected	
	0+	0+	0	0+	true
	1+	0+	1+	0+	true
	--	--	--	--	false
	0	0+	1+	0+	error
	--	--	--	--	unknown
	--	--	--	--	not evaluated
	--	--	--	--	not applicable

Enumeration Value	Number of Individual Item Status Values				Existence Result
at_least_one_exists	exists	does not exist	error	not collected	
	1+	0+	0+	0+	true
	0	1+	0	0	false
	0	0+	1+	0+	error
	0	0+	0	1+	unknown
	--	--	--	--	not evaluated
	--	--	--	--	not applicable

Enumeration Value	Number of Individual Item Status Values				Existence Result
none_exist	exists	does not exist	error	not collected	
	0	0+	0	0	true
	1+	0+	0+	0+	false
	0	0+	1+	0+	error
	0	0+	0	1+	unknown
	--	--	--	--	not evaluated
	--	--	--	--	not applicable

Enumeration Value	Number of Individual Item Status Values				Existence Result
only_one_exists	exists	does not exist	error	not collected	
	1	0+	0	0	true

2+	0+	0+	0+	false
0	0+	0	0	false
0,1	0+	1+	0+	error
0,1	0+	0	1+	unknown
--	--	--	--	not evaluated
--	--	--	--	not applicable

5.3.2.2 Check Evaluation

Check Evaluation is the process of determining whether or not the number of collected OVAL Items, specified by the check property, match the specified OVAL States. The check property specifies how many of the collected OVAL Items must match the specified OVAL States in order for the OVAL Test to evaluate to *'true'*. For additional information on how to determine if the check property is satisfied, see Section 5.3.6.1 Check Enumeration Evaluation.

5.3.2.3 State Operator Evaluation

State Operator Evaluation is the process of combining the individual results, from the comparison of an OVAL Item to the specified OVAL States, according to the `state_operator` property, to produce a result for the OVAL Test. For additional information on how to determine the final result using the `state_operator` property, see Section 5.3.6.2 Operator Enumeration Evaluation.

5.3.2.4 Determining the Final OVAL Test Evaluation Result

While the final result of the OVAL Test Evaluation is the combination of the results from the three evaluations (Existence Check Evaluation, Check Evaluation, and State Operator Evaluation), how the result is calculated will vary depending upon if the optional collected object section is present in the OVAL System Characteristics. However, in either case, if the result of the Existence Check Evaluation is *'false'*, the Check and State Operator Evaluations can be ignored and the final result of the OVAL Test will be *'false'*.

5.3.2.4.1 Final OVAL Test Evaluation Result without a Collected Objects Section

When the Collected Objects section is not present in the OVAL System Characteristics, all OVAL Items present in the OVAL System Characteristics must be examined. Each OVAL Item MUST be examined to determine which match the OVAL Object according to Section 5.3.3.1 Matching an OVAL Object to an OVAL Item and Section 5.3.3.2 Matching an OVAL Object Entity to an OVAL Item Entity. Once the set of matching OVAL Items is determined, they can undergo the three different evaluations that make up OVAL Test Evaluation.

5.3.2.4.2 Final OVAL Test Evaluation Result with a Collected Objects Section

When the Collected Objects section is present in the OVAL System Characteristics the flag value of an OVAL Object, in the Collected Objects section, must be examined before the Existence Check Evaluation is performed.

If the OVAL Object, referenced by an OVAL Test, cannot be found in the Collected Objects section, the final result of the OVAL Test MUST be *'unknown'*.

Otherwise, if the OVAL Object, referenced by an OVAL Test, is found, the following guidelines must be followed when determining the final result of an OVAL Test.

- If the flag value is *'error'*, the final result of the OVAL Test MUST be *'error'*.
- If the flag value is *'not collected'*, the final result of the OVAL Test MUST be *'unknown'*.
- If the flag value is *'not applicable'*, the final result of the OVAL Test MUST be *'not applicable'*.
- If the flag value is *'does not exist'*, the final result is determined solely by performing the Check Existence Evaluation.
- If the flag value is *'complete'*, the final result is determined by first performing the Check Existence Evaluation followed by the Check Evaluation and State Operator Evaluation.
- If the flag value is *'incomplete'*, the final result is determined as follows:
    - If the `check_existence` property has a value of *'none_exist'* and one or more OVAL Items, referenced by the OVAL Object, have a status of *'exists'*, the final result of the OVAL Test MUST be *'false'*.
    - If the `check_existence` property has a value of '*only one exists'* and more than one OVAL Item, referenced by the OVAL Object, has a status of *'exists'*, the final result of the OVAL Test MUST be *'false'*.
    - If the result of the Existence Check Evaluation is true, the following special cases during the Check Evaluation MUST be considered:
        - If the Check Evaluation evaluates to *'false'*, the final result of the OVAL Test MUST be *'false'*.
        - If the `check` property has a value of *'at least one satisfies'* and the check evaluation evaluates to *'true'*, the final result of the OVAL Test MUST be *'true'*.
    - Otherwise, the final result of the OVAL Test MUST be *'unknown'*.

Enumeration Value	Test Result
error	error
complete	depends on check_existence and check attributes
incomplete	depends on check_existence and check attributes
does not exist	depends on check_existence and check attributes
not collected	unknown
not applicable	not applicable

5.3.2.5 Variable Instance

When an OVAL Test makes use of an OVAL Variable, either directly or indirectly, OVAL Test is evaluated once for each collection of values assigned to the OVAL Variable. Each evaluation result for the OVAL Tests MUST be differentiated by incrementing the `variable_instance` property once for each assigned collection of values for the OVAL Variable. When more than one collection of values is assigned to an OVAL Variable, an OVAL Test will appear in the `tests` section once for each assigned collection of values.

5.3.3 OVAL Object Evaluation

At the highest level, OVAL Object Evaluation is the process of collecting OVAL Items based on the constraints specified by the OVAL Object Entities and OVAL Behaviors, if present, in an OVAL Object. An OVAL Object contains the minimal number of OVAL Object Entities needed to uniquely identify the system state information that makes up the corresponding OVAL Item. The methodology used to collect the system state information for the OVAL Items is strictly an implementation detail. Regardless of the chosen methodology, the same OVAL Items MUST be collected on a system for a given OVAL Object except when the `flag` for the collected OVAL Object has a value of *'incomplete'*.

5.3.3.1 Matching an OVAL Object to an OVAL Item

An OVAL Item matches an OVAL Object only if every OVAL Object Entity, as guided by any OVAL Behaviors, matches the corresponding OVAL Item Entity in the OVAL Item under consideration.

5.3.3.2 Matching an OVAL Object Entity to an OVAL Item Entity

An OVAL Object Entity matches an OVAL Item Entity only if the value of the OVAL Item Entity matches the value of the OVAL Object Entity in the context of the specified datatype and operation. See Section 5.3.6.3 for additional information regarding the allowable datatypes, operations, and how they should be interpreted.

5.3.3.3 OVAL Object Entity Evaluation

OVAL Object Entity Evaluation is the process of searching for system state information that matches the values of an OVAL Object Entity in the context of the specified datatype and operation. This process is further defined below.

5.3.3.3.1 Datatype and Operation Evaluation

The datatype and operation property associated with an OVAL Object Entity specifies what system state information should be collected from the system in the form of an OVAL Item. When comparing a value specified in the OVAL Object Entity against system state information, the operation must be performed in the context of the specified datatype; the same operation for two different datatypes could yield different results. See Section 5.3.6.3 for additional information on how to apply an operation in the context of a particular datatype.

5.3.3.3.2 nil Object Entities

For many OVAL Object Entities, there are situations in which the OVAL Object Entity does not need to be considered in the evaluation of the OVAL Object. When the nil property is set to *'true'*, it indicates that the OVAL Object Entity must not be considered during OVAL Object Evaluation and must not be collected. For more information about a particular OVAL Object Entity and how the nil property affects it, see the appropriate OVAL Component Model.

5.3.3.3.3 Referencing an OVAL Variable

An OVAL Variable may be referenced from an Object Entity in order to specify multiple values or to use a value that was collected from some other source. When the `var_ref` property is specified, the `var_check` property SHOULD also be specified. See Section 5.3.6.4 Variable Check Evaluation for more information on how to evaluate an OVAL Object Entity that references a variable.

In addition to the OVAL Item Entity value matching the values specified in the OVAL Variable according to the `var_check` property, the flag associated with the OVAL Variable must also be considered. The OVAL Variable flag indicates the outcome of the collection of values for the OVAL Variable. It is important to consider this outcome because it may affect the ability of an OVAL Object Entity to successfully match the corresponding OVAL Item Entity. Additionally, this flag will also impact the collected object flag.

The following table describes what flags are valid given the flag value of the OVAL Variable referenced by an OVAL Object Entity.

Flag of OVAL Variable	Valid OVAL Object Flags
error	• *'error'*
complete	• *'error'* • *'complete'* • *'incomplete'* • *'does not exist'* • *'not collected'* • *'not applicable'*
incomplete	• *'error'* • *'incomplete'* • *'does not exist'* • *'not collected'* • *'not applicable'*
does not exist	• *'does not exist'*
not collected	• *'does not exist'*
not applicable	• *'does not exist'*

For additional information on when each flag value MUST be used, see Section 5.2.2.1.

5.3.3.3.4 Collected Object Flag Evaluation

However, when there are multiple OVAL Object Entities in an OVAL Object the flag values for each OVAL Object Entity must be considered when determining which flag values are appropriate. The following table describes how multiple flag values influence the collected object flag of the OVAL Object referencing the variable.

Resulting Flag	Number of OVAL Components with the Specified Flag					
	error	complete	incomplete	does not exist	not collected	not applicable
error	1+	0+	0+	0+	0+	0+
complete	0	1+	0	0	0	0
incomplete	0	0+	1+	0	0	0
does not exist	0	0+	0+	1+	0	0
not collected	0	0+	0+	0+	1+	0
not applicable	0	0+	0+	0+	0+	1+

5.3.3.4 Set Evaluation

The `set` construct provides the ability to combine the collected OVAL Items of one or two OVAL Objects using the set operators defined in the `SetOperatorEnumeration`. See Section 4.3.49 `SetOperatorEnumeration` for more information about the allowed set operators.

The processing of a `set` MUST be done in the following manner:

1. Identify the OVAL Objects that are part of the `set` by examining the `object_references` associated with the `set`. Each `object_reference` will refer to an OVAL Object that describes a unique set of collected OVAL Items.
2. For every defined `filter` (See Section 5.3.3.4.2 `filter`), apply the associated `filter` to each OVAL Item.
3. Apply the set operator to all OVAL Items remaining in the set.
4. The resulting OVAL Items will be the unique set of OVAL Items referenced by the OVAL Object that contains the `set`.

5.3.3.4.1 Set Operator

Set operations are used to combine multiple sets of different OVAL Items, as identified by the `object_reference` and limited by any `filter`, into a single unique set of OVAL Items. The different operators that guide process are in the `SetOperatorEnumeration`. For each operator, if only a single `object_reference` has been supplied then the resulting set is simply the complete set of OVAL Items identified by the referenced OVAL Object after any included filters have been applied.

The tables below explain how different flags are combined for each `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 represents the resulting flag when the given `set_operator` is applied.

Table 5-1 set_operator = COMPLEMENT

Enumeration Value	OVAL Object 1 Flag					
	error	complete	incomplete	does not exist	not collected	not applicable
error	error	error	error	does not exist	error	error
complete	error	complete	incomplete	does not exist	not collected	error
incomplete	error	error	error	does not exist	not collected	error
does not exist	error	complete	incomplete	does not exist	not collected	error
not collected	error	not collected	not collected	does not exist	not collected	error
not applicable	error	error	error	error	error	error

Table 5-2 set_operator = INTERSECTION

Enumeration Value	OVAL Object 1 Flag					
	error	complete	incomplete	does not exist	not collected	not applicable
error	error	error	error	does not exist	error	error
complete	error	complete	incomplete	does not exist	not collected	complete
incomplete	error	incomplete	incomplete	does not exist	not collected	incomplete
does not exist	does not exist	does not exist	does not exist	does not exist	does not exist	does not exist
not collected	error	not collected	not collected	does not exist	not collected	not collected
not applicable	error	complete	incomplete	does not exist	not collected	not applicable

Table 5-3 set_operator = UNION

Enumeration Value	OVAL Object 1 Flag					
	error	complete	incomplete	does not exist	not collected	not applicable
error	error	error	error	error	error	error
complete	error	complete	incomplete	complete	incomplete	complete
incomplete	error	incomplete	incomplete	incomplete	incomplete	incomplete
does not exist	error	complete	incomplete	does not exist	incomplete	does not exist
not collected	error	incomplete	incomplete	incomplete	not collected	not collected
not applicable	error	complete	incomplete	does not exist	not collected	not applicable

5.3.3.4.2 filter

The `filter` construct provides a way to control the OVAL Items that are included a `set`. See Section 5.3.3.5 OVAL Filter Evaluation for additional information.

5.3.3.4.3 object_reference

When evaluating an `object_reference`, an error MUST be reported it the OVAL Object identifier is invalid, the referenced OVAL Object does not exist, or the referenced OVAL Object does not align with the OVAL Object that is referring to it.

5.3.3.5 OVAL Filter Evaluation

An OVAL Filter is a mechanism that provides the capability to either include or exclude OVAL Items based on their system state information. This is done through the referencing of an OVAL State that specifies the requirements for a matching OVAL Item and the `action` property that states whether or not the matching OVAL Items will be included or excluded.

When evaluating an OVAL Filter, an error MUST be reported if the OVAL State identifier is not legal, the referenced OVAL State does not exist, or the referenced OVAL State does not align with the OVAL Object where it is used.

The `action` property specifies whether or not the matching OVAL Items will be included or excluded. The `action` property enumeration values are defined in Section 4.3.46 ArithmeticEnumeration.

5.3.3.5.1 Applying Multiple Filters

When multiple OVAL Filters are specified, they MUST be evaluated sequentially from first to last to the collection of OVAL Items under consideration.

5.3.3.6 OVAL Object Filter

When applying a filter to OVAL Objects, every collected OVAL Item is compared to the OVAL State referenced by the OVAL Filter. If the collected OVAL Items match the OVAL State they are included or excluded based on the `action` property. The final set of collected OVAL Items is the set of collected OVAL Items after each OVAL Filter is evaluated. See Section 5.3.3.5 OVAL Filer Evaluation for additional information.

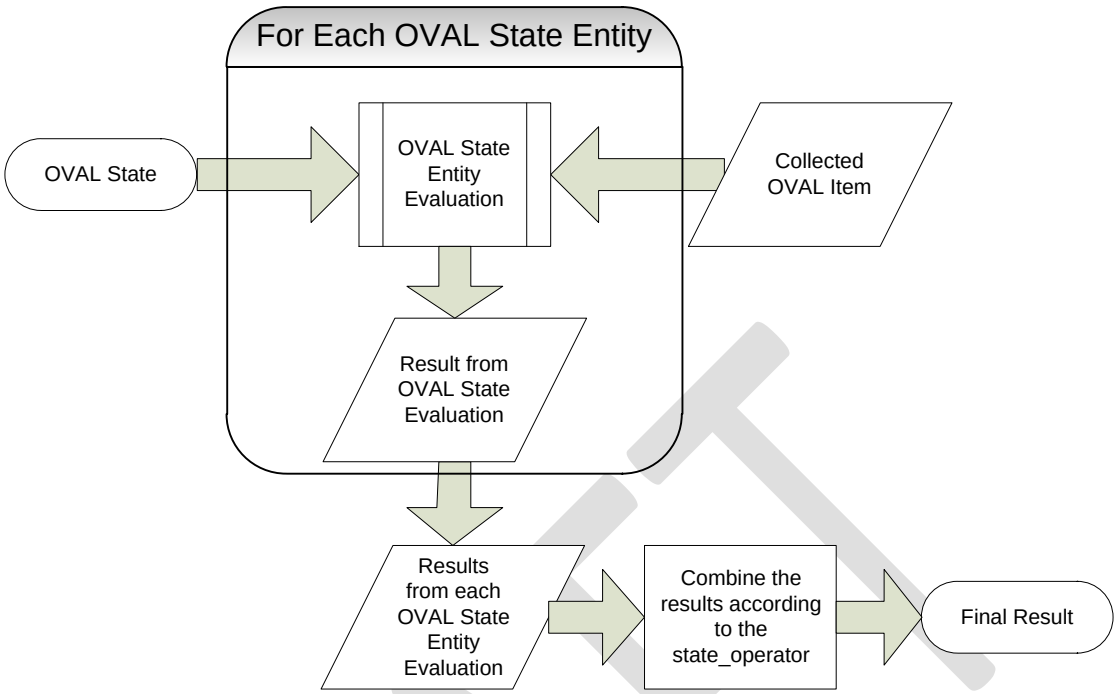
5.3.4 OVAL State Evaluation

The OVAL State is the standardized representation for expressing an expected machine state. In the OVAL State each OVAL State Entity expresses the expected value(s) for a single piece of configuration information. OVAL State Evaluation is the process of comparing a specified OVAL State against a collected OVAL Item on the system. OVAL State Evaluation can be broken up into two distinct parts:

1. **State Entity Evaluation** – The process of determining whether or not an OVAL Item Entity, in a collected OVAL Item, matches the corresponding OVAL State Entity specified in an OVAL State.

2. **State Operator Evaluation** – The process of combining the individual results, from the comparison of an OVAL Item Entity against the specified OVAL State Entity, according to the operator property.

The following diagram describes OVAL State Evaluation.



5.3.4.1 OVAL State Entity Evaluation

OVAL State Entity Evaluation is the process of comparing a specified OVAL State Entity against the corresponding collected OVAL Item Entities. This comparison must be done in the context of the datatype and operation, whether or not an OVAL Variable is referenced, and whether or not there are multiple occurrences of the corresponding OVAL Item Entity in the collected OVAL Item.

5.3.4.1.1 Datatype and Operation Evaluation

The datatype and operation property associated with an OVAL State Entity specifies how the collected OVAL Item Entity compares to the value(s) specified in the OVAL State Entity. When comparing a value specified in the OVAL State Entity against a collected OVAL Item Entity, the operation must be performed in the context of the specified datatype. See Section 5.3.3.3.1 Datatype and Operation Evaluation for additional information on how an operation is applied in the context of a particular datatype.

5.3.4.1.2 var_check Evaluation

An OVAL Variable can be referenced from an OVAL State Entity to specify multiple values that the corresponding OVAL Item Entities will be compared against or to utilize a value that was collected from some other source. For information on how to evaluate an OVAL State Entity that references an OVAL Variable, see Section 5.3.6.4 Variable Check Evaluation.

5.3.4.1.3 entity_check Evaluation

An OVAL Item may contain multiple occurrences of an OVAL Item Entity to represent that the OVAL Item has multiple values for that particular OVAL Item Entity. The `entity_check` property specifies how many occurrences of an OVAL Item Entity MUST match the OVAL State Entity, as defined in Section

5.3.4.1 OVAL State Entity Evaluation, in order to evaluate to *'true'*. The valid values for the `entity_check` property are defined by the `CheckEnumeration`. See Section 5.3.6.1 Check Enumeration Evaluation for more information about how to apply the property.

5.3.4.1.4 Determining the Final Result of an OVAL State Entity Evaluation

The final result of an OVAL State Entity Evaluation is determined by first comparing the value specified in the OVAL State Entity with each occurrence of a corresponding OVAL Item Entity, in an OVAL Item, in the context of the specified `datatype` and `operation` as defined in Section 5.3.3.3.1 Datatype and Operation Evaluation. The results of the comparisons are evaluated against the specified `entity_check` property according to Section 5.3.6.1 Check Enumeration Evaluation. This will be the final result of the OVAL State Entity Evaluation unless an OVAL Variable was also referenced.

If an OVAL Variable was referenced, the above procedure must be performed for each value in the OVAL Variable. The final result must then be computed by examining the `var_check` property and the individual results for each OVAL Variable value comparison. See Section 5.3.6.4 Variable Check Evaluation.

5.3.4.2 Operator Evaluation

Once the OVAL State Entity Evaluation is complete for every OVAL State Entity, the individual results from each evaluation MUST be combined according to the `operator` property specified on the OVAL State. The combined result will be the final result of the OVAL State Evaluation. See Section 5.3.6.2 Operator Enumeration Evaluation for more information on applying the `operator` to the individual results of the evaluations.

5.3.5 OVAL Variable Evaluation

OVAL Variable Evaluation is the process of retrieving a collection of values from sources both local and external to OVAL Definitions as well as manipulating those values through the evaluation of OVAL Functions. OVAL Variables can be used in OVAL Definitions to specify multiple values, manipulate values, retrieve values at execution time, and create generic and reusable content.

5.3.5.1 Constant Variable

A `constant_variable` is a locally defined collection of one or more values that are specified prior to evaluation time.

5.3.5.1.1 Determining the Flag Value

A `constant_variable` is only capable of having a flag value of *'error'*, *'complete'*, or *'not collected'*. The flag value of *'does not exist'* is not used for the evaluation of a `constant_variable` because a constant variable is required to contain at least one value. The following table outlines when a constant variable will evaluate to each of the flag values.

FlagEnumeration Value	Description
error	This flag value must be used when one or more values do not conform to the

	specified datatype as defined in the `oval:DatatypeEnumeration`.
complete	This flag value must be used when all values conform to the specified datatype and the collection of constant variables is supported in the OVAL-capable product.
incomplete	-
does not exist	-
not collected	-
not applicable	-

5.3.5.2 External Variable

An `external_variable` is a locally declared, externally defined, collection of one or more values. The values referenced by an `external_variable` are collected from the external source at run-time.

5.3.5.2.1 Validating External Variable Values

The OVAL Language provides the `PossibleValueType` and `PossibleRestriction` constructs as a mechanism to validate input coming from sources external to the OVAL Definitions.

5.3.5.2.1.1 Possible Restriction

The `possible_restriction` construct specifies one or more restrictions on the values of an external variable. When more than one restriction is used the individual results of each comparison between the restriction and the external variable value must be combined using the selected `operator` attribute. The default operation performed is '*AND*'. See Section 5.3.6.2 Operator Enumeration Evaluation for more information on how to combine the individual results. The final result, after combining the individual results, will be the result of the `possible_restriction` construct.

5.3.5.2.1.1.1 Restriction

Each restriction allows for the specification of an operation and a value that will be compared to a supplied value for the `external_variable`. The result of this comparison will be used in the computation of the final result of the `possible_restriction` construct. See Section 5.3.5.2.1.3 for additional information on how to determine the result of the comparison between the specified value and the external variable value using the specified operation in the context of the datatype specified on the `external_variable`.

5.3.5.2.1.2 Possible Value

The `possible_value` construct specifies a permitted external variable value. The specified value and the external variable value must be compared as string values using the equals operation. See Section 5.3.5.2.1.3 for additional information on how to determine the result of the comparison. The result of this comparison will be used in determining the final result of validating an external variable value.

5.3.5.2.1.3 Determining the Final Result of Validating an External Variable Value

The final result of validating an `external_variable` value is determined by combining every `possible_restriction` and `possible_value` constructs using the logical *'OR'* operator. That is, each value in the `external_variable` will be evaluated against the combination of

possible_restriction and possible_value constructs and the results of this evaluation will be combined using the *'OR'* operator. See Section 5.3.9.2 Operator Enumeration Evaluation for more information on how to combine the individual results using the *'OR'* operator.

5.3.5.2.2 Determining the Flag Value

An external variable is only capable of returning a flag value of *'error'*, *'complete'*, *'does not exist'*, or *'not collected'*. The following table outlines when an external variable will evaluate to each of the flag values.

FlagEnumeration Value	Description
error	This flag value must be used when one or more values do not conform to the specified datatype as defined in the oval:DatatypeEnumeration. This flag value must be used when there was an error collecting the values from the external source. This flag value must be used when there is a value, collected from the external source, that does not conform to the restrictions specified by the possible_value and possible_restriction constructs or if there is an error processing the possible_value and possible_restriction constructs. This flag value must be used when the final result of validating the external variable values is not *'true'*. This flag must be used when the external source for the variable cannot be found.
complete	This flag value must be used when the final result of validating every external variable value is *'true'* and conforms to the specified datatype.
incomplete	-
does not exist	-
not collected	-
not applicable	-

5.3.5.3 Local Variable

A local_variable is a locally defined collection of one or more values that may be composed of values from other sources collected at evaluation time.

5.3.5.3.1 OVAL Function Evaluation

An OVAL Function is a construct, in the OVAL Language, that takes one or more collections of values and manipulates them in some defined way. The result of evaluating an OVAL Function will be zero or more values.

5.3.5.3.1.1 Nested Functions

Due to the recursive nature of the ComponentGroup construct, OVAL Functions can be nested within one another. In this case, a depth-first approach is taken to processing OVAL Functions. As a result, the

inner most OVAL Functions are evaluated first, and then the resulting values are used as input to the outer OVAL Function and so on.

5.3.5.3.1.2 Evaluating OVAL Functions with Sub-components with Multiple Values

When one or more of the specified sub-components resolve to multiple values, the function will be applied to the Cartesian product[13] of the values, in the sub-components, and will result in a collection of values.

5.3.5.3.1.3 Casting the Input of OVAL Functions

OVAL Functions are designed to work on values with specific datatypes. If an input value is encountered that does not align with required datatypes an attempt must be made to cast the input value(s) to the required datatype before evaluating the OVAL Function. If the input value cannot be cast to the required datatype the flag value, of the OVAL Function, MUST be set to *'error'*.

5.3.5.3.1.4 Determining the Flag Value

When determining the flag value of an OVAL Function, the combined flag value of the sub-components must be computed in order to determine if the evaluation of the OVAL Function should continue. The following tables outline how to combine the sub-component flag values.

Notation	Description
X	*x* individual OVAL Component flag values are…
x, *y*	*x* or *y* individual OVAL Component flag values are…
x+	*x* or more individual OVAL Component flag values are…

Resulting Flag	Number of OVAL Components with the Specified Flag					
	error	complete	incomplete	does not exist	not collected	not applicable
error	1+	0+	0+	0+	0+	0+
complete	0	1+	0	0	0	0
incomplete	0	0+	1+	0	0	0
does not exist	0	0+	0+	1+	0	0
not collected	0	0+	0+	0+	1+	0
not applicable	0	0+	0+	0+	0+	1+

[13] Cartesian Product http://en.wikipedia.org/wiki/Cartesian_product

Once the flag values of the sub-components have been combined the evaluation of an OVAL Function must only continue if the flag value is *'complete'*. All other flag values mean that the evaluation of the OVAL Function stops and the flag of the OVAL Function MUST be *'error'*. The following table outlines how to determine the flag value of an OVAL Function.

FlagEnumeration Value	Description
error	This flag value must be used if the combined sub-component flag is a value other than *'complete'*. This flag value must be used if an error occurred during the computation of an OVAL Function. This flag value must be used if an attempt to cast an input value to a required datatype failed.
complete	This flag value must be used if the combined sub-component flag is complete and the evaluation of the OVAL Function completes successfully.
incomplete	-
does not exist	-
not collected	-
not applicable	-

5.3.5.3.2 OVAL Components

A component is a reference to another part of the content that allows further evaluation or manipulation of the value or values specified by the referral.

5.3.5.3.2.1 Literal Component

A `literal_component` is a component that allows the specification of a literal value. The value can be of any supported datatype as specified in the `oval:DatatypeEnumeration`. The default datatype is *'string'*.

5.3.5.3.2.1.1 Determining the Flag Value

A `literal_component` is only capable of evaluating to a flag value of *'error'* or *'complete'*. The following table outlines when a `literal_component` will evaluate to each of the flag values.

FlagEnumeration Value	Description
error	This flag value must be used when the value does not conform to the specified datatype as defined in the `oval:DatatypeEnumeration`.
complete	This flag value must be used when the value conforms to the specified datatype as defined in the `oval:DatatypeEnumeration`.
incomplete	-
does not exist	-
not collected	-

not applicable	-

5.3.5.3.2.2 Object Component

An object component is a component that resolves to the value(s) of OVAL Item Entities or OVAL Fields, in OVAL Items, that were collected by an OVAL Object. The property, `object_ref`, must reference an existing OVAL Object.

The value that is used by the object component must be specified using the `item_field` property of the object component. This indicates which entity should be used as the value for the component. In the case that the OVAL Object collects multiple OVAL Items as part of its evaluation, this can resolve to a collection of values. In the case that an OVAL Item Entity has a datatype of *'record',* the `record_field` property can be used to indicate which field to use for the component.

5.3.5.3.2.2.1 Determining the Flag Value

An `object_component` is only capable of evaluating to a flag value of *'error'*, *'complete'*, *'incomplete'*, or *'not collected'*. The following table outlines when an `object_component` will evaluate to each of the flag values.

FlagEnumeration Value	Description
error	This flag value must be used when the value does not conform to the specified datatype as defined in the `oval:DatatypeEnumeration`. This flag value must be used if the OVAL Object does not return any OVAL Items. This flag value must be used if an entity is not found with a name that matches the value of the `item_field` property. This flag value must be used if a field is not found with a name that matches the value of the `record_field` property.
complete	This flag value must be used when every value conforms to the specified datatype as defined in the `oval:DatatypeEnumeration` and when the flag of the referenced OVAL Object is *'complete'*.
incomplete	This flag value must be used when every value conforms to the specified datatype as defined in the `oval:DatatypeEnumeration` and when the flag of the referenced OVAL Object is *'incomplete'*.
does not exist	-
not collected	This flag value must be used when the OVAL-capable product does not support the collection of `object_components`.
not applicable	-

5.3.5.3.2.3 Variable Component Flag Value

A `variable_component` is only capable of evaluating to a flag value of *'error'*, *'complete'*, *'incomplete'*, or *'not collected'*. The following table outlines when a `variable_component` will evaluate to each of the flag values.

FlagEnumeration Value	Description
error	This flag value must be used when the flag value of the referenced OVAL Variable is *'error'*. This flag value must be used when the referenced OVAL Variable cannot be found.
complete	This flag value must be used when the flag value of the referenced OVAL Variable is *'complete'*.
incomplete	This flag value must be used when the flag value of the referenced OVAL Variable is *'incomplete'*.
does not exist	This flag value must be used when the flag value of the referenced OVAL Variable is *'does not exist'*.
not collected	This flag value must be used when the OVAL-capable product does not support the collection of `variable_components`.
not applicable	-

5.3.5.3.3 Determining the Flag Value

A `local_variable` can contain an OVAL Function or an OVAL Component. As a result, the flag value must consider both the flag of the OVAL Function or OVAL Component along with the additional conditions from being an OVAL Variable. The following table describes when each flag value must be used.

FlagEnumeration Value	Description
error	This flag value must be used when one or more values do not conform to the specified datatype as defined in the `oval:DatatypeEnumeration`. This flag value must be used when there was an error collecting the values from the external source. This flag value must be used when the specified datatype is *'record'*. This flag value must be used when the flag value of the specified OVAL Function or OVAL Component is *'error'*.
complete	This flag value must be used when the flag value of the specified OVAL Function or OVAL Component is *'complete'* and every value conforms to the specified datatype.
incomplete	-
does not exist	This flag value must be used when there are no values.
not collected	This flag value must be used when the OVAL-capable product does not support

	the collection of `local_variables`.
not applicable	-

5.3.6 Common Evaluation Concepts

This section describes a set of evaluation concepts that apply to several aspects of producing OVAL Content.

5.3.6.1 Check Enumeration Evaluation

Check Enumeration Evaluation is the process of determining whether or not the number of individual results, produced from the comparison of some set of values, satisfies the specified `CheckEnumeration` value.

The following tables describe how each `CheckEnumeration` value affects the final result of an evaluation. The far left column identifies the `CheckEnumeration` value in question. The middle column specifies the different combinations of individual results that the `CheckEnumeration` value may bind together. The last column specifies the final result according to each combination of individual results. It is important to note that if an individual result is negated, then a '*true*' result is '*false*' and a '*false*' result is '*true*', and all other results stay as is.

Enumeration Value	Number of Individual Results						Final Result
all	true	false	error	unknown	not evaluated	not applicable	
	1+	0	0	0	0	0+	true
	0+	1+	0+	0+	0+	0+	false
	0+	0	1+	0+	0+	0+	error
	0+	0	0	1+	0+	0+	unknown
	0+	0	0	0	1+	0+	not evaluated
	0	0	0	0	0	1+	not applicable

Enumeration Value	Number of Individual Results						Final Result
at least one	true	false	error	unknown	not evaluated	not applicable	
	1+	0+	0+	0+	0+	0+	true
	0	1+	0	0	0	0+	false
	0	0+	1+	0+	0+	0+	error
	0	0+	0	1+	0+	0+	unknown
	0	0+	0	0	1+	0+	not evaluated
	0	0	0	0	0	1+	not applicable

Enumeration Value	Number of Individual Results						Final Result
none satisfy	true	false	error	unknown	not evaluated	not applicable	
	0	1+	0	0	0	0+	true
	1+	0+	0+	0+	0+	0+	false
	0	0+	1+	0+	0+	0+	error

	0	0+	0	1+	0+	0+	unknown
	0	0+	0	0	1+	0+	not evaluated
	0	0	0	0	0	1+	not applicable

Enumeration Value	Number of Individual Results						Final Result
only one	true	false	error	unknown	not evaluated	not applicable	
	1	0+	0	0	0	0+	true
	2+	0+	0+	0+	0+	0+	false
	0	1+	0	0	0	0+	
	0,1	0+	1+	0+	0+	0+	error
	0,1	0+	0	1+	0+	0+	unknown
	0,1	0+	0	0	1+	0+	not evaluated
	0	0	0	0	0	1+	not applicable

5.3.6.2 Operator Enumeration Evaluation

Operator Enumeration Evaluation is the process of combining the individual results of evaluations using logical operations. The following table shows the notation used when describing the number of individual results that evaluate to a particular result.

Notation	Description
X	x individual results are…
x, y	x or y individual results are…
x+	x or more individual results are…
Odd	an odd number of individual results are…
Even	an even number of individual results are…

The following tables describe how each `OperatorEnumeration` value affects the final result of an evaluation. The far left column identifies the `OperatorEnumeration` value in question. The middle column specifies the different combinations of individual results that the `OperatorEnumeration` value may bind together. The last column specifies the final result according to each combination of individual results. It is important to note that if an individual result is negated, then a '*true*' result is '*false*' and a '*false*' result is '*true*', and all other results stay as is.

Enumeration Value	Number of Individual Results						Final Result
AND	true	false	error	unknown	not evaluated	not applicable	
	1+	0	0	0	0	0+	true
	0+	1+	0+	0+	0+	0+	false
	0+	0	1+	0+	0+	0+	error
	0+	0	0	1+	0+	0+	unknown
	0+	0	0	0	1+	0+	not evaluated
	0	0	0	0	0	1+	not applicable

Enumeration Value	Number of Individual Results						Final Result
ONE	true	false	error	unknown	not evaluated	not applicable	
	1+	0+	0	0	0	0+	true
	2+	0+	0+	0+	0+	0+	false
	0	1+	0	0	0	0+	false
	0,1	0+	1+	0+	0+	0+	error
	0,1	0+	0	1+	0+	0+	unknown
	0,1	0+	0	0	1+	0+	not evaluated
	0	0	0	0	0	1+	not applicable

Enumeration Value	Number of Individual Results						Final Result
OR	true	false	error	unknown	not evaluated	not applicable	
	1+	0+	0+	0+	0+	0+	true
	0	1+	0	0	0	0+	false
	0	0+	1+	0+	0+	0+	error
	0	0+	0	1+	0+	0+	unknown
	0	0+	0	0	1+	0+	not evaluated
	0	0	0	0	0	1+	not applicable

Enumeration Value	Number of Individual Results						Final Result
XOR	true	false	error	unknown	not evaluated	not applicable	
	odd	0+	0	0	0	0+	true
	even	0+	0	0	0	0+	false
	0+	0+	1+	0+	0+	0+	error
	0+	0+	0	1+	0+	0+	unknown
	0+	0+	0	0	1+	0+	not evaluated
	0	0	0	0	0	1+	not applicable

5.3.6.3 OVAL Entity Evaluation

OVAL Entity Evaluation is the process of comparing the specified value(s), from an OVAL Object or State Entity, against the corresponding system state information in the context of the selected datatype and operation.

5.3.6.3.1 Datatype and Operation Evaluation

The result of applying an operation in the context of a specified datatype MUST evaluate to *'true'* only if the values being compared satisfy the conditions of the operation for the specified datatype. If the values being compared do not satisfy the conditions of the operation, the final result MUST be *'false'*.

To ensure consistency in the comparison of the value(s) specified in the OVAL Object and State Entities with the system state information, the operations for each datatype must be defined. The following table describes how each operation must be performed in the context of a specific datatype.

<table>
<tr><th>Enumeration Value</th><th>Description of Operations</th></tr>
<tr><td>binary</td><td>Data of this type conforms to the W3C Recommendation for hex-encoded binary data [1].

equals: The collected binary value is equal to the specified binary value only if the collected binary value and the specified binary value are the same length and the collected binary value and the specified binary value contain the same characters in the same positions.

not equal: The collected binary value is not equal to the specified binary value only if the collected binary value is not the same length as the specified binary value or the collected binary value and specified binary value do not contain the same characters in the same positions.</td></tr>
<tr><td>boolean</td><td>Data of this type conforms to the W3C Recommendation for boolean data [2].

equals:
<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="2">Collected Value</th></tr>
<tr><th>false / 0</th><th>true / 1</th></tr>
<tr><td rowspan="2">Specified Value</td><td>false / 0</td><td>true</td><td>false</td></tr>
<tr><td>true / 1</td><td>false</td><td>true</td></tr>
</table>
not equal:
<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="2">Collected Value</th></tr>
<tr><th>false / 0</th><th>true / 1</th></tr>
<tr><td rowspan="2">Specified Value</td><td>false / 0</td><td>false</td><td>true</td></tr>
<tr><td>true / 1</td><td>true</td><td>false</td></tr>
</table></td></tr>
<tr><td>evr_string</td><td>Data of this type conforms to the format EPOCH:VERSION-RELEASE and comparisons involving this type MUST follow the algorithm described in the rpmVersionCompare() function which is located in lib/psm.c of the RPM source code.

equals: The collected evr_string value c is equal to the specified evr_string value s only if the result of the algorithm described in the rpmVersionCompare(c,s) function is 0.

not equal: The collected evr_string value c is not equal to the specified evr_string value s only if the result of the algorithm described in the rpmVersionCompare(c,s) function is -1 or 1.

greater than: The collected evr_string value c is greater than the specified evr_string s value only if the result of the algorithm described in the rpmVersionCompare(c,s)</td></tr>
</table>

	function is 1. **greater than or equal:** The collected evr_string value *c* is greater than or equal to the specified evr_string value *s* only if the result of the algorithm described in the rpmVersionCompare(*c,s*) function is 1 or 0. **less than:** The collected evr_string value *c* is less than the specified evr_string value *s* only if the result of the algorithm described in the rpmVersionCompare(*c,s*) function is -1. **less than or equal:** The collected evr_string value *c* is less than or equal to the specified evr_string value *s* only if the result of the algorithm described in the rpmVersionCompare(*c,s*) function is -1 or 0.
fileset_revision	Data of this type conforms to the version string related to filesets in HP-UX. An example would be 'A.03.61.00'. *Please note that this needs further community review and discussion.*
float	Data of this type conforms to the W3C Recommendation for float data [3]. **equals:** The collected float value is equal to the specified float value only if the collected float value and the specified float value are numerically equal. **not equal:** The collected float value is not equal to the specified float value only if the collected float value and the specified float value are not numerically equal. **greater than:** The collected float value is greater than the specified float value only if the collected float value is numerically greater than the specified float value. **greater than or equal:** The collected float value is greater than or equal to the specified float value only if the collected float value is numerically greater than or equal to the specified float value. **less than:** The collected float value is less than the specified float value only if the collected float value is numerically less than the specified float value. **less than or equal:** The collected float value is less than or equal to the specified float value only if the collected float value is numerically less than or equal to the specified float value.
ios_version	Data of this type conforms to Cisco IOS Train strings. These are in essence version strings for IOS. Please refer to Cisco's IOS Reference Guide for information on how to compare different Trains as they follow a very specific pattern.[17] *Please note that this needs further community review and discussion.*

int	Data of this type conforms to the W3C Recommendation for integer data [4]. **equals:** The collected integer value is equal to the specified integer value only if the collected integer value and the specified integer value are numerically equal. **not equal:** The collected integer value is not equal to the specified integer value only if the collected integer value and the specified integer value are not numerically equal. **greater than:** The collected integer value is greater than the specified integer value only if the collected integer value is numerically greater than the specified integer value. **greater than or equal:** The collected integer value is greater than or equal to the specified integer value only if the collected integer value is numerically greater than or equal to the specified integer value. **less than:** The collected integer value is less than the specified integer value only if the collected integer value is numerically less than the specified integer value. **less than or equal:** The collected integer value is less than or equal to the specified integer value only if the collected integer value is numerically less than or equal to the specified integer value. **bitwise and:** The collected integer satisfies the bitwise and operation with the specified integer value only if the result of performing the bitwise and operation on the binary representation of the collected integer value and the binary representation of the specified integer value is the binary representation of the specified value. **bitwise or:** The collected integer satisfies the bitwise or operation with the specified integer value only if the result of performing the bitwise or operation on the binary representation of the collected integer value and the binary representation of the specified integer value is the binary representation of the specified value.
ipv4_address	The ipv4_address datatype represents IPv4 addresses and IPv4 address prefixes. Its value space consists of the set of ordered pairs of integers where the first element of each pair is in the range [0,2^32) (the representable range of a 32-bit unsigned int), and the second is in the range [0,32]. The first element is an address, and the second is a prefix length. The lexical space is dotted-quad CIDR-like notation ('a.b.c.d' where 'a', 'b', 'c', and 'd' are integers from 0-255), optionally followed by a slash ('/') and either a prefix length (an integer from 0-32) or a netmask represented in the dotted-quad notation described previously. Examples of legal values are '192.0.2.0', '192.0.2.0/32', and '192.0.2.0/255.255.255.255'. Additionally, leading zeros are permitted such that '192.0.2.0' is equal to '192.000.002.000'. If a prefix length is not specified, it is

	implicitly equal to 32. All operations are defined in terms of the value space. Let A and B be ipv4_address values (i.e. ordered pairs from the value space). The following definitions assume that bits outside the prefix have been zeroed out. By zeroing the low order bits, they are effectively ignored for all operations. Implementations of the following operations MUST behave as if this has been done. Let P_addr mean the first element of ordered pair P and P_prefix mean the second element. **equals:** A equals B if and only if A_addr == B_addr and A_prefix == B_prefix. **not equal:** A is not equal to B if and only if they don't satisfy the criteria for operator "equals". **greater than:** A is greater than B if and only if A_prefix == B_prefix and A_addr > B_addr. If A_prefix != B_prefix, i.e. prefix lengths are not equal, an error MUST be reported. **greater than or equal:** A is greater than or equal to B if and only if A_prefix == B_prefix and they satisfy either the criteria for operators "equal" or "greater than". If A_prefix != B_prefix, i.e. prefix lengths are not equal, an error MUST be reported. **less than:** A is less than B if and only if A_prefix == B_prefix and they don't satisfy the criteria for operator "greater than or equal". If A_prefix != B_prefix, i.e. prefix lengths are not equal, an error MUST be reported. **less than or equal:** A is less than or equal to B if and only if A_prefix == B_prefix and they don't satisfy the criteria for operator "greater than". If A_prefix != B_prefix, i.e. prefix lengths are not equal, an error MUST be reported. **subset of:** A is a subset of B if and only if every IPv4 address in subnet A is present in subnet B. In other words, A_prefix >= B_prefix and the high B_prefix bits of A_addr and B_addr are equal. **superset of:** A is a superset of B if and only if B is a subset of A.
ipv6_address	The ipv6_address datatype represents IPv6 addresses and IPv6 address prefixes. Its value space consists of the set of ordered pairs of integers where the first element of each pair is in the range [0,2^128) (the representable range of a 128-bit unsigned int), and the second is in the range [0,128]. The first element is an address, and the second is a prefix length. The lexical space is CIDR notation given in IETF specification RFC 4291 for textual representations of IPv6 addresses and IPv6 address prefixes (see sections 2.2 and 2.3). If a prefix-length is not specified, it is implicitly equal to 128.

	All operations are defined in terms of the value space. Let A and B be ipv6_address values (i.e. ordered pairs from the value space). The following definitions assume that bits outside the prefix have been zeroed out. By zeroing the low order bits, they are effectively ignored for all operations. Implementations of the following operations MUST behave as if this has been done. Let P_addr mean the first element of ordered pair P and P_prefix mean the second element. **equals:** A equals B if and only if A_addr == B_addr and A_prefix == B_prefix. **not equal:** A is not equal to B if and only if they don't satisfy the criteria for operator "equals". **greater than:** A is greater than B if and only if A_prefix == B_prefix and A_addr > B_addr. If A_prefix != B_prefix, an error MUST be reported. **greater than or equal:** A is greater than or equal to B if and only if A_prefix == B_prefix and they satisfy either the criteria for operators "equal" or "greater than". If A_prefix != B_prefix, an error MUST be reported. **less than:** A is less than B if and only if A_prefix == B_prefix and they don't satisfy the criteria for operator "greater than or equal". If A_prefix != B_prefix, an error MUST be reported. **less than or equal:** A is less than or equal to B if and only if A_prefix == B_prefix and they don't satisfy the criteria for operator "greater than". If A_prefix != B_prefix, an error MUST be reported. **subset of:** A is a subset of B if and only if every IPv6 address in subnet A is present in subnet B. In other words, A_prefix >= B_prefix and the high B_prefix bits of A_addr and B_addr are equal. **superset of:** A is a superset of B if and only if B is a subset of A.
string	Data of this type conforms to the W3C Recommendation for string data [6]. **equals:** The collected string value is equal to the specified string value only if the collected string value and the specified string value are the same length and the collected string value and the specified string value contain the same characters in the same positions. **not equal:** The collected string value is not equal to the specified string value only if

	the collected string value is not the same length as the specified string value or the collected string value and specified string value do not contain the same characters in the same positions. **case insensitive equals**: The collected string value is equal to the specified string value only if the collected string value and the specified string value are the same length and the collected string value and the specified string value contain the same characters, regardless of case, in the same positions. **case insensitive not equal**: The collected string value is not equal to the specified string value only if the collected string value and the specified string value are not the same length or the collected string value and the specified string value do not contain the same characters, regardless of case, in the same positions. **pattern match**: The collected string value will match the specified string value only if the collected string value matches the specified string value when the specified string is interpreted as a Perl 5 Compatible Regular Expression (PCRE)[9].
version	Data of this type represents a value that is a hierarchical list of non-negative integers separated by a single character delimiter. Any single non-integer character may be used as a delimiter and the delimiter may vary between the non-negative integers of a given version value. The hierarchical list of non-negative integers must be compared sequentially from left to right. When the version values, under comparison, have different-length lists of non-negative integers, zeros must be appended to the end of the values such that the lengths of the lists of non-negative integers are equal. **equals:** The collected version value is equal to the specified version value only if every non-negative integer in the collected version value is numerically equal to the corresponding non-negative integer in the specified version value. **not equal:** The collected version value is not equal to the specified version value if any non-negative integer in the collected version value is not numerically equal to the corresponding non-negative integer in the specified version value. **greater than:** The collected version value *c* is greater than the specified version value *s* only if the following algorithm returns true: $c = c_1,c_2,...,c_n$ where , is any non-integer character $s = s_1,s_2,...,s_n$ where , is any non-integer character for i = 1 to n if $c_i > s_i$ return true if $c_i < s_i$ return false if $c_i == s_i$

	if *i* != *n* continue else return false **greater than or equal:** The collected version value *c* is greater than or equal to the specified version value *s* only if the following algorithm returns true: $c = c_1,c_2,...,c_n$ where , is any non-integer character $s = s_1,s_2,...,s_n$ where , is any non-integer character for *i* = 1 to *n* if $c_i > s_i$ return true if $c_i < s_i$ return false if $c_i == s_i$ if *i* != *n* continue else return true **less than:** The collected version value *c* is less than the specified version value *s* only if the following algorithm returns true: $c = c_1,c_2,...,c_n$ where , is any non-integer character $s = s_1,s_2,...,s_n$ where , is any non-integer character for *i* = 1 to *n* if $c_i < s_i$ return true if $c_i > s_i$ return false if $c_i == s_i$ if *i* != *n* continue else return false **less than or equal:** The collected version value *c* is less than or equal to the specified version value *s* only if the following algorithm returns true: $c = c_1,c_2,...,c_n$ where , is any non-integer character $s = s_1,s_2,...,s_n$ where , is any non-integer character for *i* = 1 to *n*

	if $c_i < s_i$ return true if $c_i > s_i$ return false if $c_i == s_i$ if i != n continue else return true
record	Data of this type describes an entity with structured set of named fields and values as its content. The record datatype is currently prohibited from being used on variables. **equals**: The collected record value is equal to the specified record value only if each specified OVAL Field has a corresponding collected OVAL Field with the same name property and that the collected OVAL Field value matches the specified OVAL Field value in the context of the datatype and operation as described above.

5.3.6.4 Variable Check Evaluation

It is often necessary to reference a variable from an OVAL Object or State Entity in order to specify multiple values or to use a value that was collected at runtime. When an OVAL Variable is referenced from an OVAL Object or State Entity using the `var_ref` property, the system state information will be compared to the every OVAL Variable value in the context of the specified datatype and operation. The final result of these comparisons are dependent on the value of the `var_check` property which specifies how many of the values, contained in OVAL Variable, must match the system state information to evaluate to a result of *'true'*. The valid values for the `var_check` property are the defined in the `CheckEnumeration`.

Enumeration Value	Description
all	The OVAL Object or State Entity matches the system state information only if the value of the OVAL Item Entity matches all of the values in the referenced the OVAL Variable in the context of the datatype and operation specified in the OVAL Object or State Entity.
at least one	The OVAL Object or State Entity matches the system state information only if the value of the OVAL Item Entity matches one or more of the values in the referenced OVAL Variable in the context of the datatype and operation specified in the OVAL Object or State Entity.
none satisfy	The OVAL Object or State Entity matches the system state information only if the OVAL Item Entity matches zero of the values in the referenced OVAL Variable in the context of the specified datatype and operation.
only one	The OVAL Object or State Entity matches the system state information only if the OVAL Item Entity matches one of the values in the referenced OVAL Variable in the context of the specified datatype and operation.

5.3.6.4.1 *Determining the Final Result of the Variable Check Evaluation*

For more detailed information on how to combine the individual results of the comparisons between the OVAL object or State Entities and the system state information to determine the final result of applying the `var_check` property, see Section 5.3.6.1 Check Enumeration Evaluation.

5.3.6.5 *OVAL Entity Casting*

In certain situations, it is possible that the datatype specified on the OVAL Entity is different from the datatype of the system state information. When this happens, it is required that an attempt is made to cast the system state information to the datatype specified by the OVAL Entity before the operation is applied. If the cast is unsuccessful, the final result of the OVAL Entity Evaluation MUST be *'error'*. Otherwise, the final result is dependent on the outcome of the Datatype and Operation Evaluation and the Variable Check Evaluation if an OVAL Variable is referenced. The process of casting a value of one datatype to a value of another datatype must conform to Section 5.3.8 Entity Casting.

5.3.7 Masking Data

When the `mask` property is set to *'true'* on an OVAL Entity or an OVAL Field, the value of that OVAL Entity or OVAL Field MUST NOT be present in the OVAL Results. Additionally, the `mask` property MUST be set to *'true'* for any OVAL Entity or OVAL Field or corresponding OVAL Item Entity or OVAL Field in the OVAL Results where the system state information was omitted.

When the `mask` property is set to *'true'* on an OVAL Entity with a `datatype` of *'record'*, each OVAL Field MUST have its operation and value or value omitted from the OVAL Results regardless of the OVAL Field's mask property value.

It is possible for masking conflicts to occur where one entity has `mask` set to *'true'* and another entity has `mask` set to *'false'*. Such a conflict will occur when the `mask` attribute is set differently on an OVAL Object and OVAL State or when more than one OVAL Objects identify the same OVAL Item(s). When such a conflict occurs the value MUST always be masked.

Values MUST NOT be masked in OVAL System Characteristics that are not contained within OVAL Results.

5.3.8 Entity Casting

Casting is performed whenever the datatype of a value, used during evaluation, differs from the specified datatype whether it be on an OVAL Entity or an OVAL Function. In most scenarios, it will be possible to attempt the cast of a value from one datatype to another.

5.3.8.1 *Attempting to Cast a Value*

When attempting to cast a value from one datatype to another, the value under consideration must be parsed according to the specified datatype. If the value is successfully parsed according to the definition of the specified datatype in the oval:DatatypeEnumeration, this constitutes a successful cast. If the value is not successfully parsed according to the definition of the specified datatype, this means that it is

not possible to cast the value to the specified datatype and an error MUST be reported for the construct attempting to perform the cast.

5.3.8.2 Prohibited Casting

In some scenarios, it is not possible to perform a cast from one datatype to another due to the datatypes, under consideration, being incompatible. When an attempt is made to cast two incompatible datatypes, an error MUST be reported. The following outlines the casts where the datatypes are incompatible:

- An attempt to cast a value of datatype *'record'* to any datatype other than *'record'*.
- An attempt to cast a value of datatype *'ipv4_address'* to any datatype other than *'ipv4_address'* or *'string'*.
- An attempt to cast a value of datatype *'ipv6_address'* to any datatype other than *'ipv6_address'* or *'string'*.
- An attempt to cast a value with a datatype other than *'ipv4_address'* or *'string'* to *'ipv4_address'*.
- An attempt to cast a value with a datatype other than *'ipv6_address'* or *'string'* to *'ipv6_address'*.

6 XML Representation

The XML Representation for the OVAL Language Data Model is documented via a series of XML Schemas.[14] These schemas describe how the information presented in this Specification is formatted and represented as XML Documents. Please refer to the appropriate Schema for more information about a specific XML representation.

OVAL Common Model
https://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/oval-common-schema.xsd

OVAL Definitions Model
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/oval-definitions-schema.xsd

OVAL System Characteristics Model
http://oval.mitre.org/language/version5.10.1/ovalsc/complete/oval-system-characteristics-schema.xsd

[14] XML Schema Part 0: Primer Second Edition http://www.w3.org/TR/xmlschema-0/

OVAL Results Model
http://oval.mitre.org/language/version5.10.1/ovalresults/complete/oval-results-schema.xsd

OVAL Variables Model
http://oval.mitre.org/language/version5.10.1/ovalvar/complete/oval-variables-schema.xsd

OVAL Directives Model
http://oval.mitre.org/language/version5.10.1/ovaldir/complete/oval-directives-schema.xsd

The complete listing of XML representation resources can be found on the OVAL website.[15]

6.1 Signature Support

In order to ensure integrity and authenticity of content, the OVAL Data Model supports the use of XML Digital Signatures as defined by the W3C. These signatures can be used to provide confidence that the data and intent of OVAL Content has not been compromised or modified from its original state.

XML Digital Signatures may be applied to the entire collection of content at once or to the individual pieces of the content such as OVAL Definitions, OVAL Tests, OVAL Objects, etc.

OVAL uses an enveloped XML Digital Signature as described in the official *XML Digital Signatures Specification* as defined by the W3C. For more information, please refer to the *XML Signature Syntax and Processing Specification*.[16]

6.2 XML Extensions

In a number of locations in the OVAL XML Schemas, extension points exist to allow additional XML fragments to be provided as part of the XML Document. These extension points are implemented using the xsd:any[17] element. They are included in the OVAL Language to facilitate experimentation, in the form of adding additional information within the XML Document, and to allow vendors and content authors to provide details that are not currently part of the OVAL Language.

The xsd:any construct allows the addition of any valid XML within OVAL content. Unlike the other content allowed in OVAL, this content is not constrained by the OVAL Language schema.

6.3 ElementMapType

The `ElementMapType` explicitly states the OVAL Object, OVAL State, and OVAL Item associated with a specific OVAL Test in the OVAL Language. This mapping allows tools to programmatically determine this information at run-time. Within the OVAL Language XML Schema representation this type is used by

[15] See the OVAL Language documentation at: http://oval.mitre.org/language/version5.10.1/
[16] XML Signature Syntax and Processing Specification http://www.w3.org/TR/xmldsig-core/
[17] XML Schema Definition of the Any Element, Any Attribute http://www.w3.org/TR/xmlschema-0/#any

each OVAL Test in the various OVAL Component Models. The `ElementMapType` MUST not be used in OVAL Content.

Property	Type	Multiplicity	Description
test	string	1	The name of the OVAL Test being mapped.
object	string	0..1	The OVAL Object associated with the specified OVAL Test.
state	string	0..1	The OVAL State associated with the specified OVAL Test.
Item	string	0..1	The OVAL Item associated with the specified OVAL Test.

6.4 Official OVAL Component Models

Below is a list of the current, official OVAL Component Models:

AIX
Defines tests targeted for IBM's AIX Operating System platform.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/aix-definitions-schema.xsd

Apache
[Deprecated] Defines tests targeted for Apache web server software.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/apache-definitions-schema.xsd

Cisco CatOS
Defines tests targeted for Cisco's CatOS platform, used for network switches.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/catos-definitions-schema.xsd

VMWare ESX
Defines tests targeted for VMWare's ESX Server platform.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/esx-definitions-schema.xsd

FreeBSD
Defines tests targeted for the FreeBSD Operating System platform.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/freebsd-definitions-schema.xsd

HP-UX
Defines tests targeted for Hewlett-Packard's HP-UX Operating System platform.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/hpux-definitions-schema.xsd

Independent
Defines tests that are independent of a specific software platform.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/independent-definitions-schema.xsd

Cisco IOS
Defines tests targeted for Cisco's IOS platform, used for network switches and routers.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/ios-definitions-schema.xsd

Linux
Defines tests targeted for a broad set of LINUX-based Operating System platforms.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/linux-definitions-schema.xsd

MacOS
Defines tests targeted for Apple's MacOS Operating System platform.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/macos-definitions-schema.xsd

Cisco PixOS
Defines tests targeted for Cisco's Pix OS platform, used for IP firewalls and NAT appliances.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/pixos-definitions-schema.xsd

Microsoft SharePoint
Defines tests targeted for Microsoft's SharePoint software.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/sharepoint-definitions-schema.xsd

Solaris
Defines tests targeted for the Solaris Operating System platform.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/solaris-definitions-schema.xsd

UNIX
Defines tests targeted for a broad set of UNIX-based Operating System platforms.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/unix-definitions-schema.xsd

Windows
Defines tests targeted for the Microsoft Windows Operating System platform.
http://oval.mitre.org/language/version5.10.1/ovaldefinition/complete/windows-definitions-schema.xsd

6.5 Use of xsi:nil

When authoring OVAL Content, it is sometimes required or desirable to make use of an OVAL Entity that contains no content. This can even apply to entities whose XML Schema indicates that they should have content. Within OVAL, entities that are allowed to be "nillable" by their XML Schema can use the @xsi:nil attribute to indicate that the entity should have no content associated with it.

The interpretation or meaning of an entity that has @xsi:nil="true" set is dependent on the meaning assigned to the entity by the appropriate documentation. Any entity that allows an @xsi:nil attribute to be set must define how this case should be interpreted.

6.6 Validation Requirements

All XML content written against the XML Schema implementation of the OVAL Language MUST be both XML Schema and Schematron valid as defined in the XML Schemas associated with the XML Schema implementation of the OVAL Language.

Appendix A – Extending the OVAL Language Data Model

The OVAL Language Data Model defines a set of core capabilities, as described within this Specification document, with numerous extension points. This appendix highlights the opportunities for extension with in the OVAL Language. It is important to understand the role of OVAL Component Models within the OVAL Language, as they allow OVAL to easily expand to new platforms and system constructs. Additionally, this appendix will raise awareness of the other extension points that have been built into the OVAL Language.

OVAL Component Models

The core capabilities described above establish a framework for defining OVAL Tests that are related at some level by the software they describe. Tests that are identical across multiple platforms, and thus represent a more general class of tests, are grouped together in an OVAL Component Model.

These platform-specific constructs are defined in their own Models, called OVAL Component Models. The OVAL Component Models each provide the necessary constructs (i.e., OVAL Tests, OVAL Objects, and OVAL States) to accomplish checks that apply to the given platform.

When considering a new OVAL Component Model, it is important to understand what commonality will be captured by the new extension. Additionally, the low-level APIs and other relevant implementation information should be understood in order to confirm that viability of the implementation of the extension.

Within the OVAL Component Models, similar concepts or concepts that are related to a type of platform are grouped together. These groupings are purely conceptual, as there is no actual linking between them. An author's OVAL Definitions can pull content from multiple different OVAL Component Models. This structure allows the ability to group checks that relate to a broad section of software together, while still retaining the ability to separate disparate ones.

OVAL Definitions Model

The following sections describe how the OVAL Definitions Model is extended by OVAL Component Models to develop platform specific constructs in the OVAL Language.

New OVAL Tests

OVAL Tests serve as the mechanism for combining an OVAL Object with one or more OVAL States. When creating an OVAL Component Model, a test is created that extends the abstract OVAL Definitions Model `TestType` construct.

An OVAL Test extension will typically define the specific OVAL Object and OVAL State that are combined to form the OVAL Test extension. Additionally the extension will provide documentation regarding the extension that describes its purpose and use. All of the remaining detail (the relevant data that must be collected and how to evaluate the check) will be part of the OVAL Object and/or OVAL State.

New OVAL Objects

OVAL Objects describe the system-level detail that is required for completing the check that is being defined. Within an OVAL Component Model, an object is created to capture the required information by extending the abstract OVAL Definitions Model `oval-def:ObjectType` construct.

OVAL Objects typically align with low-level system APIs or other system level structures. This allows those implementing the new OVAL Object to more easily understand how to access the required information while executing the assessment.

In order to provide the required information for an OVAL Object extension, the construct needs to provide documentation for the extension, and also add any required OVAL Entities to capture the necessary data for the check. For information on adding OVAL Entities, see Section 0 New OVAL Entity.

Optionally, the OVAL Object can define a set of behaviors. These behaviors are used to better direct one or more aspects of how the required data is collected for the entity.

New OVAL States

OVAL States describe the necessary conditions under which a collected OVAL Item should be considered a passing check. As such, within an OVAL Component Model a state is created to capture the required information by extending the abstract OVAL Definitions Model `StateType` construct.

In order to provide the required information for an OVAL State extension, the construct needs to provide documentation for the extension and also add any required entities to capture the information that will determine the result of the check.

Additionally, an OVAL State extension typically requires that all of the entities that were defined as part of the associated OVAL Object extension are also included in the OVAL State extension.

New OVAL Entity

An OVAL Entity represents a single piece of system configuration, and is used by OVAL Objects, OVAL States, and OVAL Items, each with slightly different meaning. When used in the context of an OVAL Object the OVAL Entity provides a way to uniquely identify a single OVAL Item. When used in the context of an OVAL State the OVAL Entity provides a way to specify the expected value(s) of an OVAL Item. When used in the context of an OVAL Item, the OVAL Entity indicates a property and its value that has been collected.

When creating an OVAL Entity, the following pieces of information need to be defined:

- datatype
- operation restrictions when used in an OVAL Object or OVAL State
- use of xsi:nil

The datatype can be any of the datatypes defined by the OVAL Language. The operation restrictions refer to any limitations on the allowed operations for a specific OVAL Entity. The superset of available

operations is determined by the `oval:OperationEnumeration`. A restriction can be added in the OVAL Component Model to limit the available operations to a subset of the enumeration.

Additionally, any OVAL entity that allows the use of nil must define what meaning that condition has when used. See Section 6.5 Use of xsi:nil.

OVAL System Characteristics Model

New OVAL Items

OVAL Items describe the system-level details that have been collected as part of an assessment. As such, within an OVAL Component Model an item is created to capture the collected information by extending the abstract OVAL System Characteristics Model `ItemType` construct.

In order to provide the required information for an OVAL Item extension, the construct needs to provide documentation for the extension as well as all of the entities that need to exist to hold all of the collected item's relevant information.

Extension Points within the OVAL Definitions Model

In addition to the OVAL Component Models, other extension points exist within the OVAL Definitions Model. Those additional extension points are described here.

Generator Information

The `generator` construct captures information about the author or tool that created the content found in the current context. It allows extension via an xsd:any value, which lets an author or tool provide additional XML information regarding the content's creation.

For more information about xsd:any usage, see Section 6.2 XML Extensions.

OVAL Definition Metadata

The Metadata content provides additional contextual information regarding the OVAL Content. It captures information such as title, description, and affected platform and product information. Additionally, the Metadata can provide additional information using the xsd:any construct.

For more information about xsd:any usage, see Section 6.2 XML Extensions.

Extension Points within the OVAL System Characteristics Model

The OVAL System Characteristics Model provides the framework capabilities for detailing the information that has been collected as part of an assessment. To provide a way to communicate these details for a given low-level, this model is extended in the two ways, Generator Information and System Information.

Generator Information

The `generator` construct captures information about the author or tool that created the content found in the current context. It allows extension via an xsd:any value, which lets an author or tool provide additional XML information regarding the content's creation.

For more information about xsd:any usage, see Section 6.2 XML Extensions.

System Information

The `system_information` construct provides detailed information about the system that has been targeted for assessment. It captures information such as the host name, the IP address for the target, and other relevant asset-related fields.

Additionally, the information here can be extended using the xsd:any construct to provide additional asset-related information.

Integrating Asset Identification

The Asset Identification specification[18] provides a standardized way of reporting asset information across different organizations. Asset Identification elements can hold data useful for identifying what tool, what version of that tool was used, and identify other assets used to compile an OVAL document (e.g. persons, organizations, etc.).

To support greater interoperability, the `oval-sc:system_info` property supports an extension point that allows arbitrary data to be supplied. It is at this point that OVAL content MAY make use of extensions to provide AI information using the AI Specification. Please see the Extensions Section: 6.2 XML Extensions for more information.

For more information about xsd:any usage, see Section 6.2 XML Extensions.

OVAL Results Model

The OVAL Results Model provides the framework capabilities for communicating the results of an assessment. This model may be extended through Generator Information.

Generator Information

The `generator` construct captures information about the author or tool that created the content found in the current context. It allows extension via an xsd:any value, which lets an author or tool provide additional XML information regarding the content's creation.

For more information about xsd:any usage, see Section 6.2 XML Extensions.

[18] Asset Inventory (AI): http://scap.nist.gov/specifications/ai/

Appendix B - OVAL Language Versioning Policy

The OVAL Language Versioning Policy is used to determine whether a new revision will require a major version change, minor version change, or a version update, and how version information is represented and conveyed in the OVAL Language.

A three-component version identifier is used to track the evolution of the OVAL Language over time. Each component of the version identifier is a numeric value and corresponds to one of the three release types — "Major", "Minor", and "Update" — each of which is subject to the OVAL Language Revision Policy. The complete version identifier has the following form: MAJOR.MINOR.UPDATE. For example, "5.10.1".

A high-level overview of each type of OVAL release is described below:

- Major Release – A major release is for adding features that require breaking backward compatibility with previous versions of the OVAL Language or represent fundamental changes to concepts in the OVAL Language.
- Minor Release – A minor release is for adding features that do not break backward compatibility with previous versions of the OVAL Language.
- Update Release – An update release is reserved for fixing critical defects in a particular version of the OVAL Language that affects the usability of the release.

The complete *OVAL Language Versioning Policy* is available on the OVAL website.[19]

Appendix C - OVAL Language Deprecation Policy

When an OVAL Language construct is marked as deprecated its usage becomes strongly discouraged and it will be removed in a later release. Constructs may be removed for a number of reasons including security issues, language consistency, or obsolescence. When a language construct is deprecated in remains as a valid construct of the OVAL Language for at least one release cycle of the OVAL Language. All deprecated constructs are clearly annotated in the OVAL Language schemas and this specification document including a detailed description of the justification for deprecation.

The complete *OVAL Language Deprecation Policy* is available on the OVAL website.[20]

[19] The OVAL Language Versioning Policy https://oval.mitre.org/language/about/versioning.html
[20] The OVAL Language Deprecation Policy http://oval.mitre.org/language/about/deprecation.html

Appendix D - Regular Expression Support

The OVAL Language supports a common subset of the regular expression character classes, operations, expressions, and other lexical tokens defined within Perl 5's regular expression specification. This common subset was identified through a survey of several regular expression libraries in an effort to ensure that the regular expression elements supported by OVAL will be compatible with a wide variety of regular expression libraries. A listing of the surveyed regular expression libraries is provided later in this document.

Supported Regular Expression Syntax

Perl regular expression modifiers (m, i, s, x) are not supported. These modifiers should be considered to always be 'OFF' unless specifically permitted by documentation on an OVAL Language construct.

Character matching assumes a Unicode character set. Note that no syntax is supplied for specifying code points in hex; actual Unicode characters must be used instead.

The following regular expression elements are specifically identified as supported in the OVAL Language. For more detailed definitions of the regular expression elements listed below, refer to their descriptions in the *Perl 5.004 Regular Expression* documentation. A copy of this documentation has been preserved for reference purposes [10]. Regular expression elements that are not listed below should be avoided as they are likely to be incompatible or have different meanings with commonly used regular expression libraries.

Please note that while only a subset of the Perl 5 regular expression syntax is supported, content can be written that may still run in some OVAL interpreter tools. This practice should be avoided in order to maintain the portability of content across multiple tools. In the event that an attempt was made to evaluate a string against a malformed regular expression, an error must be reported. An example of a malformed regular expression is the pattern "+". An unsupported regular expression should only be reported as an error if the evaluating tool is not capable of analyzing the pattern. A malformed regular expression may remain ignored if the preceding existence check can determine the evaluation flag.

Metacharacters

```
\   Quote the next metacharacter
^   Match the beginning of the line
.   Match any character (except newline)
$   Match the end of the line (or before newline at the end)
|   Alternation
()  Grouping
[]  Character class
```

Greedy Quantifiers

```
*      Match 0 or more times
+      Match 1 or more times
?      Match 1 or 0 times
{n}    Match exactly n times
{n,}   Match at least n times
```

```
{n,m}  Match at least n but not more than m times
```

Reluctant Quantifiers

```
*?     Match 0 or more times
+?     Match 1 or more times
??     Match 0 or 1 time
{n}?   Match exactly n times
{n,}?  Match at least n times
{n,m}? Match at least n but not more than m times
```

Escape Sequences

```
\t     tab                   (HT, TAB)
\n     newline               (LF, NL)
\r     return                (CR)
\f     form feed             (FF)
\033   octal char (think of a PDP-11)
\x1B   hex char
\c[    control char
```

Character Classes

```
\w  Match a "word" character (alphanumeric plus "_")
\W  Match a non-word character
\s  Match a whitespace character
\S  Match a non-whitespace character
\d  Match a digit character
\D  Match a non-digit character
```

Zero Width Assertions

```
\b  Match a word boundary
\B  Match a non-(word boundary)
```

Extensions

```
(?:regexp)  - Group without capture
(?=regexp)  - Zero-width positive lookahead assertion
(?!regexp)  - Zero-width negative lookahead assertion
```

Version 8 Regular Expressions

```
[chars]  - Match any of the specified characters
[^chars] - Match anything that is not one of the specified characters
[a-b]    - Match any character in the range between "a" and "b", inclusive
a|b      - Alternation; match either the left side of the "|" or the right
side
\n       - When 'n' is a single digit: the nth capturing group matched.
```

Appendix E – Normative References

[1] W3C Recommendation for Hex-Encoded Binary Data
http://www.w3.org/TR/xmlSchema-2/#hexBinary

[2] W3C Recommendation for Boolean Data
http://www.w3.org/TR/xmlSchema-2/#boolean

[3] W3C Recommendation for Float Data
http://www.w3.org/TR/xmlSchema-2/#float

[4] W3C Recommendation for Integer Data
http://www.w3.org/TR/xmlSchema-2/#integer

[5] RFC 4291 - IP Version 6 Addressing Architecture
http://www.ietf.org/rfc/rfc4291.txt

[6] W3C Recommendation for String Data
http://www.w3.org/TR/xmlSchema-2/#string

[7] IEEE Std 802-2001 – IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture
http://standards.ieee.org/getieee802/download/802-2001.pdf

[8] Lexicographic Equality
http://www.gnu.org/software/guile/manual/html_node/String-Comparison.html

[9] Perl Compatible Regular Expression Support in OVAL
http://oval.mitre.org/language/about/re_support_5.6.html

[10] Perl 5.004 Regular Expressions
http://oval.mitre.org/language/about/perlre.html

[13] W3C Recommendation for Double Data
http://www.w3.org/TR/xmlschema-2/#double

[14] W3C Recommendation for URI Data
http://www.w3.org/TR/xmlschema-2/#anyURI

[15] W3C Recommendation for unsigned int Data
http://www.w3.org/TR/xmlschema-2/#unsignedInt

[16] RFC 2119 – Key words for use in RFCs to Indicate Requirement Levels
http://www.ietf.org/rfc/rfc2119.txt

[17] Cisco IOS Reference Manual
http://www.cisco.com/web/about/security/intelligence/ios-ref.html

[18] RFC 4632 - Classless Inter-domain Routing (CIDR)
http://tools.ietf.org/html/rfc4632

[19] RFC 791 – IPv4 Protocol Specification
http://tools.ietf.org/html/rfc791

[20] Microsoft Windows File Time Format
http://msdn.microsoft.com/en-us/library/ms724290(v=vs.85).aspx

[21] RFC 4291
http://tools.ietf.org/html/rfc4291

Appendix F - Change Log

Version 5.11 Revision 5 – December 18, 2014

- Updated version and date information for the Official 5.11 Release.

Version 5.11 Revision 4 – December 01, 2014

- Updated version and date information for 5.11 Release Candidate 2.

Version 5.11 Revision 3 – November 18, 2014

- Updated version and date information for 5.11 Release Candidate 1.

Version 5.11 Revision 2 – September 25, 2013

- Added operator attribute to PossibleRestrictionType and updated documentation (Section 4.3.25, 5.3.5.2.1.1).
- Fixed a few typos that were reported on the oval-developer-list (http://making-security-measurable.1364806.n2.nabble.com/OVAL-Language-Specification-01-20-2012-Typos-tp7580573.html).
- Changed Section 4.3.34 to say "collection" of values instead of "set".
- Fixed a typo in the title of reference [17] and updated the link (Appendix E).
- Changed occurrences of “malware and threat indicator sharing” to “malware artifact hunting” (Table of Contents and Section 2.6).
- Updated ipv4_address and ipv6_address datatype documentation based on community discussion (Section 4.2.5 and 5.3.6.3.1).
- Improved documentation for the process of validating an external_variable value (Section 5.3.5.2.1.3)
- Made explicit the Perl 5 regular expression metacharacters which must be escaped for the escape_regex and regex_capture functions. Clarified documentation regarding allowed sub-components and multi-valued components (Section 4.3.40, 4.3.45).
- Added documentation to regex_capture function to correct the expected behavior when handling zero or more than one capturing sub-pattern (Section 4.3.45).

Documented how to handle invalid and unsupported regular expression syntax (Appendix D).**Version 5.11 Revision 1 – February 20, 2013**

- Fixed the documentation describing how the equals operation should behave for the record datatype (Section 5.3.6.3.1). This addresses https://github.com/OVALProject/Language/issues/8.
- Added documentation stating that field names, for record entities in OVAL Objects and OVAL States, MUST be unique (Section 4.3.62 and 4.3.78). This addresses https://github.com/OVALProject/Language/issues/3.
- Updated version and date information for 5.11 Draft 1.

Version 5.10.1 Revision 1 – January 20, 2012

- Added documentation to explicitly state that an empty string value is allowed for entity types where it was previously implied because the only restriction on the value is that it is a string. (Section 4.3.53-60, 4.3.65-76, 4.5.15-22, and 4.5.25-28)
- Added documentation explicitly stating that an empty string value MUST be used when referencing an OVAL Variable from an OVAL Object Entity, Object Field Entity, State Entity, or State Field Entity and that an empty string value SHOULD be used when a status other than 'exists' is specified on an OVAL Item Entity or Item Field Entity. (Section 4.3.51, 4.3.62, 4.3.63, 4.3.78, 4.5.13, and 4.5.23)
- Updated the text regarding the OVAL Language Versioning Policy to reflect the change to a three-component version identifier. (Appendix B – OVAL Language Versioning Policy).
- Defined what an OVAL Item is. (Appendix G – Terms)

Version 5.10 Revision 1 – September 14, 2011

- Published initial revision of the version 5.10 specification.

Appendix G - Terms and Acronyms

Terms

OVAL Behavior – An action that can further specify the set of OVAL Items that matches an OVAL Object.

OVAL Test – An OVAL Test is the standardized representation of an assertion about the state of a system.

OVAL Object – An OVAL Object is a collection of OVAL Object Entities that can uniquely identify a single OVAL Item on the system.

OVAL Item – An OVAL Item is a single piece of collected system state information.

OVAL Component – An OVAL Construct that is specified in the `oval-def:ComponentGroup`.

OVAL Function – An OVAL Function is a capability used in OVAL Variables to manipulate a variable's value.

OVAL Variable – An OVAL Variable represents a collection of values that allow for dynamic substitutions and reuse of system state information.

OVAL Object Entity – An OVAL Object Entity is a standardized representation for specifying a single piece of system state information.

OVAL State Entity – An OVAL State Entity is a standardized representation for checking a single piece of system state information.

OVAL Item Entity – An OVAL Item Entity is a standardized representation for a single piece of system state information.

OVAL-capable product – Any product that implements one or more OVAL Adoption Capabilities as defined in the OVAL Adoption Program.

OVAL Adoption Program – An on-going effort to educate vendors on best practices regarding the use and implementation OVAL, to provide vendors with an opportunity to make formal self-assertions about how their products utilize OVAL, and to help MITRE gain deeper insights into how OVAL is or could be utilized so that the standard can continue to evolve as needed by the community.

OVAL Adoption Capability – A specific function or functions of a product, service, or repository that implements some defined aspect of the OVAL Language. The following OVAL Adoption Capabilities are currently defined as follows:

- **Authoring Tool** – A product that aids in the process of creating new OVAL files (including products that consolidate existing definitions into a single file).
- **Definition Evaluator** – A product that uses an OVAL Definition to guide evaluation and produces OVAL Results (full results) as output.
- **Definition Repository** – A repository of OVAL Definitions made available to the community (free or pay).
- **Results Consumer** – A product that accepts OVAL Results as input and either displays those results to the user, or uses the results to perform some action.
- **System Characteristics Producer** – A product that generates a valid OVAL System Characteristics file based on the details of a system.

Acronyms

CCE Common Configuration Enumeration
CPE Common Platform Enumeration
CVE Common Vulnerabilities and Exposures
DHS Department of Homeland Security
DNS Domain Name System
IP Internet Protocol
MAC Media Access Control
NAC Network Access Control
NIST National Institute of Standards and Technology
NSA National Security Agency
OVAL Open Vulnerability and Assessment Language

SIM Security Information Management
UML Unified Modeling Language
URI Uniform Resource Identifier
URN Uniform Resource Name
W3C World Wide Web Consortium
XML eXtensible Markup Language