Static Analysis Results Interchange Format (SARIF) Version 2.0

Working Draft 02

16 March 2018

Technical Committee:

[OASIS Static Analysis Results Interchange Format (SARIF) TC](https://www.oasis-open.org/committees/sarif/)

Chairs:

David Keaton ([dmk@dmk.com](mailto:dmk@dmk.com)), Individual Member

Luke Cartey ([luke@semmle.com](mailto:luke@semmle.com)), [Semmle](http://www.semmle.com/)

Editors:

Michael Fanning ([mikefan@microsoft.com](mailto:mikefan@microsoft.com)), [Microsoft](http://www.microsoft.com/)

Laurence J. Golding ([larrygolding@comcast.net](mailto:larrygolding@comcast.net)), Individual Member

Additional artifacts:

This prose specification is one component of a Work Product that also includes:

* JSON schemas:
  + sarif-schema.json
* (**Note:** Any normative computer language definitions that are part of the Work Product, such as XML instances, schemas and Java(TM) code, including fragments of such, must be (a) well formed and valid, (b) provided in separate plain text files, (c) referenced from the Work Product; and (d) where any definition in these separate files disagrees with the definition found in the specification, the definition in the separate file prevails. Remove this note before submitting for publication.)

Related work:

This specification replaces or supersedes:

* None

This specification is related to:

* None

Declared XML namespaces:

* None

Abstract:

This document defines a standard format for the output of static analysis tools. The format is referred to as the “Static Analysis Results Interchange Format”, and is abbreviated as SARIF.

Status:

This [Working Draft](https://www.oasis-open.org/policies-guidelines/oasis-defined-terms-2017-05-26#dWorkingDraft) (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or [approved](https://www.oasis-open.org/policies-guidelines/tc-process#committeeDraft) as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document [Approval Process](https://www.oasis-open.org/policies-guidelines/tc-process#standApprovProcess) begins officially with a TC vote to approve a WD as a Committee Draft. A TC may approve a Working Draft, revise it, and re-approve it any number of times as a Committee Draft.

This Working Draft is being developed under the [RF on RAND Terms](https://www.oasis-open.org/policies-guidelines/ipr#RF-on-RAND-Mode) Mode of the [OASIS IPR Policy](https://www.oasis-open.org/policies-guidelines/ipr), the mode chosen when the Technical Committee was established. All members of the TC should be familiar with this document, which may create obligations regarding the disclosure and availability of a member's patent, copyright, trademark and license rights that read on an approved OASIS specification. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC’s web page (<https://www.oasis-open.org/committees/sarif/ipr.php>).

Any machine-readable content ([Computer Language Definitions](https://www.oasis-open.org/policies-guidelines/tc-process#wpComponentsCompLang)) declared Normative for this Work Product must also be provided in separate plain text files. In the event of a discrepancy between such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

URI patterns:

Initial publication URI:  
http://docs.oasis-open.org/sarif/sarif/v1.0/csd01/sarif-v1.0-csd01.docx

Permanent “Latest version” URI:  
http://docs.oasis-open.org/sarif/sarif/v1.0/sarif-v1.0.docx

Copyright © OASIS Open 2017. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](https://www.oasis-open.org/policies-guidelines/ipr) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents

[1 Introduction 11](#_Toc510005927)

[1.1 IPR Policy 11](#_Toc510005928)

[1.2 Terminology 11](#_Toc510005929)

[1.3 Normative References 16](#_Toc510005930)

[1.4 Non-Normative References 17](#_Toc510005931)

[2 Conventions 18](#_Toc510005932)

[2.1 General 18](#_Toc510005933)

[2.2 Format examples 18](#_Toc510005934)

[2.3 Property notation 18](#_Toc510005935)

[2.4 Syntax notation 18](#_Toc510005936)

[3 File format 19](#_Toc510005937)

[3.1 General 19](#_Toc510005938)

[3.2 fileContent objects 19](#_Toc510005939)

[3.2.1 General 19](#_Toc510005940)

[3.2.2 text property 19](#_Toc510005941)

[3.2.3 binary property 19](#_Toc510005942)

[3.3 fileLocation objects 19](#_Toc510005943)

[3.3.1 General 19](#_Toc510005944)

[3.3.2 uri property 20](#_Toc510005945)

[3.3.3 uriBaseId property 20](#_Toc510005946)

[3.4 String properties 22](#_Toc510005947)

[3.5 Object properties 22](#_Toc510005948)

[3.6 Array properties 22](#_Toc510005949)

[3.6.1 General 22](#_Toc510005950)

[3.6.2 Array properties with unique values 22](#_Toc510005951)

[3.7 Property bags 22](#_Toc510005952)

[3.7.1 General 22](#_Toc510005953)

[3.7.2 Tags 22](#_Toc510005954)

[3.7.2.1 General 22](#_Toc510005955)

[3.7.2.2 Namespaced tags 22](#_Toc510005956)

[3.7.2.3 Tag metadata 23](#_Toc510005957)

[3.8 Date/time properties 24](#_Toc510005958)

[3.9 message objects 24](#_Toc510005959)

[3.9.1 General 24](#_Toc510005960)

[3.9.2 Plain text messages 25](#_Toc510005961)

[3.9.3 Rich text messages 25](#_Toc510005962)

[3.9.3.1 General 25](#_Toc510005963)

[3.9.3.2 Security implications 25](#_Toc510005964)

[3.9.4 Messages with placeholders 26](#_Toc510005965)

[3.9.5 Messages with embedded links 26](#_Toc510005966)

[3.9.6 Message string resources 27](#_Toc510005967)

[3.9.6.1 General 27](#_Toc510005968)

[3.9.6.2 Embedded string resource lookup procedure 28](#_Toc510005969)

[3.9.6.3 SARIF resource file lookup procedure 28](#_Toc510005970)

[3.9.6.4 SARIF resource file format 29](#_Toc510005971)

[3.9.6.4.1 General 29](#_Toc510005972)

[3.9.6.4.2 sarifLog object 29](#_Toc510005973)

[3.9.6.4.3 run object 30](#_Toc510005974)

[3.9.6.4.4 tool object 30](#_Toc510005975)

[3.9.6.4.5 resources object 30](#_Toc510005976)

[3.9.7 text property 30](#_Toc510005977)

[3.9.8 richText property 30](#_Toc510005978)

[3.9.9 messageId property 31](#_Toc510005979)

[3.9.10 richMessageId property 31](#_Toc510005980)

[3.9.11 arguments property 31](#_Toc510005981)

[3.10 sarifLog object 31](#_Toc510005982)

[3.10.1 General 31](#_Toc510005983)

[3.10.2 version property 32](#_Toc510005984)

[3.10.3 $schema property 32](#_Toc510005985)

[3.10.4 runs property 32](#_Toc510005986)

[3.11 run object 32](#_Toc510005987)

[3.11.1 General 32](#_Toc510005988)

[3.11.2 id property 32](#_Toc510005989)

[3.11.3 stableId property 33](#_Toc510005990)

[3.11.4 baselineId property 33](#_Toc510005991)

[3.11.5 automationId property 33](#_Toc510005992)

[3.11.6 architecture property 33](#_Toc510005993)

[3.11.7 tool property 33](#_Toc510005994)

[3.11.8 invocations property 34](#_Toc510005995)

[3.11.9 conversion property 34](#_Toc510005996)

[3.11.10 originalUriBaseIds property 34](#_Toc510005997)

[3.11.11 files property 35](#_Toc510005998)

[3.11.11.1 General 35](#_Toc510005999)

[3.11.11.2 Property names 35](#_Toc510006000)

[3.11.11.3 Property values 38](#_Toc510006001)

[3.11.12 logicalLocations property 39](#_Toc510006002)

[3.11.13 results property 40](#_Toc510006003)

[3.11.14 resources property 40](#_Toc510006004)

[3.11.15 richMessageMimeType property 40](#_Toc510006005)

[3.11.16 properties property 41](#_Toc510006006)

[3.12 tool object 41](#_Toc510006007)

[3.12.1 General 41](#_Toc510006008)

[3.12.2 name property 41](#_Toc510006009)

[3.12.3 fullName property 41](#_Toc510006010)

[3.12.4 semanticVersion property 41](#_Toc510006011)

[3.12.5 version property 42](#_Toc510006012)

[3.12.6 fileVersion property 42](#_Toc510006013)

[3.12.7 language property 42](#_Toc510006014)

[3.12.8 resourceLocation property 42](#_Toc510006015)

[3.12.9 sarifLoggerVersion property 43](#_Toc510006016)

[3.12.10 properties property 43](#_Toc510006017)

[3.13 invocation object 44](#_Toc510006018)

[3.13.1 General 44](#_Toc510006019)

[3.13.2 commandLine property 44](#_Toc510006020)

[3.13.3 arguments property 44](#_Toc510006021)

[3.13.4 responseFiles property 45](#_Toc510006022)

[3.13.5 attachments property 45](#_Toc510006023)

[3.13.6 startTime property 45](#_Toc510006024)

[3.13.7 endTime property 45](#_Toc510006025)

[3.13.8 exitCode property 45](#_Toc510006026)

[3.13.9 exitCodeDescription property 46](#_Toc510006027)

[3.13.10 exitSignalName property 46](#_Toc510006028)

[3.13.11 exitSignalNumber property 46](#_Toc510006029)

[3.13.12 processStartFailureMessage property 46](#_Toc510006030)

[3.13.13 toolExecutionSuccessful property 47](#_Toc510006031)

[3.13.14 machine property 47](#_Toc510006032)

[3.13.15 account property 47](#_Toc510006033)

[3.13.16 processId property 47](#_Toc510006034)

[3.13.17 fileName property 47](#_Toc510006035)

[3.13.18 workingDirectory property 47](#_Toc510006036)

[3.13.19 environmentVariables property 48](#_Toc510006037)

[3.13.20 toolNotifications property 48](#_Toc510006038)

[3.13.21 configurationNotifications property 48](#_Toc510006039)

[3.13.22 stdin, stdout, stderr, and stdoutStderr properties 49](#_Toc510006040)

[3.13.23 properties property 50](#_Toc510006041)

[3.14 attachment object 50](#_Toc510006042)

[3.14.1 General 50](#_Toc510006043)

[3.14.2 description property 50](#_Toc510006044)

[3.14.3 fileLocation property 50](#_Toc510006045)

[3.15 conversion object 51](#_Toc510006046)

[3.15.1 General 51](#_Toc510006047)

[3.15.2 tool property 51](#_Toc510006048)

[3.15.3 invocation property 51](#_Toc510006049)

[3.15.4 analysisToolLogFileLocation property 51](#_Toc510006050)

[3.16 file object 52](#_Toc510006051)

[3.16.1 General 52](#_Toc510006052)

[3.16.2 fileLocation property 52](#_Toc510006053)

[3.16.3 parentKey property 53](#_Toc510006054)

[3.16.4 offset property 53](#_Toc510006055)

[3.16.5 length property 54](#_Toc510006056)

[3.16.6 mimeType property 54](#_Toc510006057)

[3.16.7 hashes property 54](#_Toc510006058)

[3.16.8 contents property 54](#_Toc510006059)

[3.16.9 properties property 54](#_Toc510006060)

[3.17 hash object 55](#_Toc510006061)

[3.17.1 General 55](#_Toc510006062)

[3.17.2 value property 55](#_Toc510006063)

[3.17.3 algorithm property 55](#_Toc510006064)

[3.18 result object 56](#_Toc510006065)

[3.18.1 General 56](#_Toc510006066)

[3.18.2 Constraints 56](#_Toc510006067)

[3.18.3 id property 57](#_Toc510006068)

[3.18.4 ruleId property 57](#_Toc510006069)

[3.18.5 ruleKey property 57](#_Toc510006070)

[3.18.6 level property 58](#_Toc510006071)

[3.18.7 message property 59](#_Toc510006072)

[3.18.8 ruleMessageId property 60](#_Toc510006073)

[3.18.9 locations property 61](#_Toc510006074)

[3.18.10 snippet property 61](#_Toc510006075)

[3.18.11 fingerprintContributions property 61](#_Toc510006076)

[3.18.12 codeFlows property 62](#_Toc510006077)

[3.18.13 stacks property 62](#_Toc510006078)

[3.18.14 relatedLocations property 63](#_Toc510006079)

[3.18.15 suppressionStates property 63](#_Toc510006080)

[3.18.15.1 General 63](#_Toc510006081)

[3.18.15.2 suppressedInSource value 64](#_Toc510006082)

[3.18.15.3 suppressedExternally value 64](#_Toc510006083)

[3.18.16 baselineState property 64](#_Toc510006084)

[3.18.17 attachments property 65](#_Toc510006085)

[3.18.18 conversionProvenance property 65](#_Toc510006086)

[3.18.19 fixes property 65](#_Toc510006087)

[3.18.20 properties property 65](#_Toc510006088)

[3.19 analysisToolLogFileContents object 65](#_Toc510006089)

[3.19.1 General 65](#_Toc510006090)

[3.19.2 region property 66](#_Toc510006091)

[3.19.3 snippet property 66](#_Toc510006092)

[3.19.4 analysisToolLogFileLocation property 66](#_Toc510006093)

[3.20 location object 67](#_Toc510006094)

[3.20.1 General 67](#_Toc510006095)

[3.20.2 Constraints 67](#_Toc510006096)

[3.20.3 analysisTarget property 68](#_Toc510006097)

[3.20.4 resultFile property 68](#_Toc510006098)

[3.20.5 fullyQualifiedLogicalName property 68](#_Toc510006099)

[3.20.6 logicalLocationKey property 69](#_Toc510006100)

[3.20.7 decoratedName property 70](#_Toc510006101)

[3.20.8 properties property 70](#_Toc510006102)

[3.21 physicalLocation object 70](#_Toc510006103)

[3.21.1 General 70](#_Toc510006104)

[3.21.2 id property 70](#_Toc510006105)

[3.21.3 fileLocation property 71](#_Toc510006106)

[3.21.4 region property 71](#_Toc510006107)

[3.22 region object 72](#_Toc510006108)

[3.22.1 General 72](#_Toc510006109)

[3.22.2 Text regions 72](#_Toc510006110)

[3.22.3 Binary regions 73](#_Toc510006111)

[3.22.4 startLine property 74](#_Toc510006112)

[3.22.5 startColumn property 74](#_Toc510006113)

[3.22.6 endLine property 74](#_Toc510006114)

[3.22.7 endColumn property 74](#_Toc510006115)

[3.22.8 offset property 74](#_Toc510006116)

[3.22.9 length property 74](#_Toc510006117)

[3.23 logicalLocation object 75](#_Toc510006118)

[3.23.1 General 75](#_Toc510006119)

[3.23.2 name property 75](#_Toc510006120)

[3.23.3 kind property 75](#_Toc510006121)

[3.23.4 parentKey property 76](#_Toc510006122)

[3.24 codeFlow object 76](#_Toc510006123)

[3.24.1 General 76](#_Toc510006124)

[3.24.2 message property 76](#_Toc510006125)

[3.24.3 locations property 76](#_Toc510006126)

[3.24.4 properties property 76](#_Toc510006127)

[3.25 stack object 76](#_Toc510006128)

[3.25.1 General 76](#_Toc510006129)

[3.25.2 message property 77](#_Toc510006130)

[3.25.3 frames property 77](#_Toc510006131)

[3.25.4 properties property 77](#_Toc510006132)

[3.26 stackFrame object 77](#_Toc510006133)

[3.26.1 General 77](#_Toc510006134)

[3.26.2 message property 77](#_Toc510006135)

[3.26.3 physicalLocation property 77](#_Toc510006136)

[3.26.4 module property 77](#_Toc510006137)

[3.26.5 threadId property 77](#_Toc510006138)

[3.26.6 fullyQualifiedLogicalName property 77](#_Toc510006139)

[3.26.7 logicalLocationKey property 78](#_Toc510006140)

[3.26.8 address property 78](#_Toc510006141)

[3.26.9 offset property 78](#_Toc510006142)

[3.26.10 parameters property 78](#_Toc510006143)

[3.26.11 properties property 78](#_Toc510006144)

[3.27 annotatedCodeLocation object 78](#_Toc510006145)

[3.27.1 General 78](#_Toc510006146)

[3.27.2 step property 78](#_Toc510006147)

[3.27.3 physicalLocation property 79](#_Toc510006148)

[3.27.4 fullyQualifiedLogicalName property 79](#_Toc510006149)

[3.27.5 logicalLocationKey property 79](#_Toc510006150)

[3.27.6 module property 80](#_Toc510006151)

[3.27.7 threadId property 80](#_Toc510006152)

[3.27.8 message property 80](#_Toc510006153)

[3.27.9 kind property 80](#_Toc510006154)

[3.27.10 kind-dependent properties: target, targetLocation, values and state 81](#_Toc510006155)

[3.27.11 targetKey property 87](#_Toc510006156)

[3.27.12 importance property 87](#_Toc510006157)

[3.27.13 taintKind property 87](#_Toc510006158)

[3.27.14 snippet property 87](#_Toc510006159)

[3.27.15 annotations property 88](#_Toc510006160)

[3.27.16 properties property 88](#_Toc510006161)

[3.28 annotation object 88](#_Toc510006162)

[3.28.1 General 88](#_Toc510006163)

[3.28.2 message property 89](#_Toc510006164)

[3.28.3 locations property 89](#_Toc510006165)

[3.29 resources object 89](#_Toc510006166)

[3.29.1 General 89](#_Toc510006167)

[3.29.2 messageStrings property 89](#_Toc510006168)

[3.29.3 rules property 89](#_Toc510006169)

[3.30 rule object 90](#_Toc510006170)

[3.30.1 General 90](#_Toc510006171)

[3.30.2 Constraints 90](#_Toc510006172)

[3.30.3 id property 90](#_Toc510006173)

[3.30.4 name property 91](#_Toc510006174)

[3.30.5 shortDescription property 91](#_Toc510006175)

[3.30.6 fullDescription property 91](#_Toc510006176)

[3.30.7 messageStrings property 91](#_Toc510006177)

[3.30.8 richMessageStrings property 92](#_Toc510006178)

[3.30.9 helpUri property 92](#_Toc510006179)

[3.30.10 help property 92](#_Toc510006180)

[3.30.11 configuration property 92](#_Toc510006181)

[3.30.12 properties property 93](#_Toc510006182)

[3.31 ruleConfiguration object 93](#_Toc510006183)

[3.31.1 General 93](#_Toc510006184)

[3.31.2 enabled property 93](#_Toc510006185)

[3.31.3 defaultLevel property 93](#_Toc510006186)

[3.31.4 parameters property 93](#_Toc510006187)

[3.32 fix object 94](#_Toc510006188)

[3.32.1 General 94](#_Toc510006189)

[3.32.2 description property 94](#_Toc510006190)

[3.32.3 fileChanges property 94](#_Toc510006191)

[3.33 fileChange object 95](#_Toc510006192)

[3.33.1 General 95](#_Toc510006193)

[3.33.2 fileLocation property 95](#_Toc510006194)

[3.33.3 replacements property 95](#_Toc510006195)

[3.34 replacement object 95](#_Toc510006196)

[3.34.1 General 95](#_Toc510006197)

[3.34.2 Constraints 96](#_Toc510006198)

[3.34.3 deletedRegion property 97](#_Toc510006199)

[3.34.4 insertedContent property 97](#_Toc510006200)

[3.35 notification object 97](#_Toc510006201)

[3.35.1 General 97](#_Toc510006202)

[3.35.2 id property 97](#_Toc510006203)

[3.35.3 ruleId property 97](#_Toc510006204)

[3.35.4 ruleKey property 97](#_Toc510006205)

[3.35.5 physicalLocation property 98](#_Toc510006206)

[3.35.6 message property 98](#_Toc510006207)

[3.35.7 level property 98](#_Toc510006208)

[3.35.8 threadId property 99](#_Toc510006209)

[3.35.9 time property 99](#_Toc510006210)

[3.35.10 exception property 99](#_Toc510006211)

[3.35.11 properties property 99](#_Toc510006212)

[3.36 exception object 99](#_Toc510006213)

[3.36.1 General 99](#_Toc510006214)

[3.36.2 kind property 99](#_Toc510006215)

[3.36.3 message property 99](#_Toc510006216)

[3.36.4 stack property 100](#_Toc510006217)

[3.36.5 innerExceptions property 100](#_Toc510006218)

[4 Conformance 101](#_Toc510006219)

[4.1 Conformance targets 101](#_Toc510006220)

[4.2 Conformance Clause 1: SARIF log file 101](#_Toc510006221)

[4.3 Conformance Clause 2: SARIF resource file 101](#_Toc510006222)

[4.4 Conformance Clause 3: SARIF producer 101](#_Toc510006223)

[4.5 Conformance Clause 4: Direct producer 101](#_Toc510006224)

[4.6 Conformance Clause 5: Converter 102](#_Toc510006225)

[4.7 Conformance Clause 6: Deterministic producer 102](#_Toc510006226)

[4.8 Conformance Clause 7: SARIF consumer 102](#_Toc510006227)

[4.9 Conformance Clause 8: Viewer 102](#_Toc510006228)

[Appendix A. (Informative) Acknowledgments 103](#_Toc510006229)

[Appendix B. (Informative) Use of fingerprints by result management systems 104](#_Toc510006230)

[Appendix C. (Informative) Use of SARIF by log file viewers 105](#_Toc510006231)

[Appendix D. (Informative) Production of SARIF by converters 106](#_Toc510006232)

[Appendix E. (Informative) Locating rule metadata 107](#_Toc510006233)

[Appendix F. (Normative) Producing deterministic SARIF log files 108](#_Toc510006234)

[F.1 General 108](#_Toc510006235)

[F.2 Non-deterministic file format elements 108](#_Toc510006236)

[F.3 Array and dictionary element ordering 109](#_Toc510006237)

[F.4 Absolute paths 109](#_Toc510006238)

[F.5 Compensating for non-deterministic output 109](#_Toc510006239)

[F.6 Interaction between determinism and baselining 110](#_Toc510006240)

[Appendix G. (Informative) Guidance on fixes 111](#_Toc510006241)

[Appendix H. (Informative) Examples 112](#_Toc510006242)

[H.1 Minimal valid SARIF log file 112](#_Toc510006243)

[H.2 Minimal recommended SARIF log file with source information 112](#_Toc510006244)

[H.3 Minimal recommended SARIF log file without source information 113](#_Toc510006245)

[H.4 SARIF resource file with rule metadata 114](#_Toc510006246)

[H.5 Comprehensive SARIF file 115](#_Toc510006247)

[Appendix I. (Informative) Revision History 122](#_Toc510006248)

# Introduction

Software developers use a variety of analysis tools to assess the quality of their programs. These tools report results which can indicate problems related to program qualities such as correctness, security, performance, compliance with contractual or legal requirements, compliance with stylistic standards, understandability, and maintainability. To form an overall picture of program quality, developers often need to aggregate the results produced by all of these tools. This aggregation is more difficult if each tool produces output in a different format.

This document defines a standard format for the output of static analysis tools. The goals of the format are:

* Comprehensively capture the range of data produced by commonly used static analysis tools.
* Be a useful format for analysis tools to emit directly, and also an effective interchange format into which the output of any analysis tool can be converted.
* Be suitable for use in a variety of scenarios related to analysis result management, and be extensible for use in new scenarios.
* Reduce the cost and complexity of aggregating the results of various analysis tools into common workflows.
* Capture information that is useful for assessing a project’s compliance with corporate policy or certification standards.
* Adopt a widely used serialization format that can be parsed by readily available tools.
* Represent analysis results for all kinds of programming artifacts, including source code and object code.
* Represent the logical construct against which a result is produced, such as a function, class, or namespace.
* Represent the physical location at which a result is produced, including problems that are detected in nested files (such as a source file within a compressed container).

## IPR Policy

This Working Draft is being developed under the [RF on RAND Terms](https://www.oasis-open.org/policies-guidelines/ipr#RF-on-RAND-Mode) Mode of the [OASIS IPR Policy](https://www.oasis-open.org/policies-guidelines/ipr), the mode chosen when the Technical Committee was established.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC’s web page (<https://www.oasis-open.org/committees/sarif/ipr.php>).

## Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [[BCP14](#BCP14)] [[RFC2119](#RFC2119)] [[RFC8174](#RFC8174)] when, and only when, they appear in all capitals, as shown here.

For purposes of this document, the following terms and definitions apply:

analysis target

[programming artifact](#def_programming_artifact) which a [static analysis tool](#def_static_analysis_tool) is instructed to analyze

artifact

see [programming artifact](#def_programming_artifact)

baseline

set of [results](#def_result) produced by a single [run](#def_run) of a set of [static analysis tools](#def_static_analysis_tool) on a set of [programming artifacts](#def_programming_artifact)

NOTE: A [result management system](#def_result_management_system) can compare the results of a subsequent run to a baseline to determine whether new results have been introduced.

binary file

[file](#def_file) considered as a sequence of bytes

binary region

[region](#def_region) representing a contiguous range of zero or more bytes in a [binary file](#def_binary_file)

call stack

sequence of nested function calls

camelCase name

name that begins with a lowercase letter, in which each subsequent word begins with an uppercase letter

Example: camelCase, version, fullName.

code flow

sequence of program locations that specify a possible execution path through the code

column

1-based index of a character within a [line](#def_line)

converter

[SARIF producer](#def_sarif_producer) that transforms the output of an [analysis tool](#def_static_analysis_tool) from its native output format into the SARIF format

direct producer

[analysis tool](#def_static_analysis_tool) which acts as a [SARIF producer](#def_sarif_producer)

embedded link

syntactic construct which enables a [message string](#def_message_string) to refer to a location within a [file](#def_file) mentioned in a [result](#def_result)

embedded resource

[resource](#def_resource) that is contained within a [SARIF log file](#def_sarif_log_file)

empty array

array that contains no elements, and so has a length of 0

empty object

object that contains no properties

empty string

string that contains no characters, and so has a length of 0

(end) user

person who uses the information in a [log file](#def_log_file) to investigate, [triage](#def_triage), or resolve [results](#def_result)

external resource

[resource](#def_resource) that is contained within a [SARIF resource file](#def_sarif_resource_file)

false positive

[result](#def_result) which an [end user](#def_end_user) decides does not actually represent a [problem](#def_problem)

file

sequence of bytes accessible *via* a URI

Example: A physical file in a file system, a specific version of a file in a version control system.

fingerprint

[stable value](#def_stable_value) that can be used by a [result management system](#def_result_management_system) to uniquely identify a [result](#def_result) over time, even if the [programming artifact](#def_programming_artifact) in which it occurs is modified

line

contiguous sequence of characters, starting either at the beginning of a [file](#def_file) or immediately after a [newline sequence](#def_newline_sequence), and ending at and including the nearest subsequent newline sequence, if one is present, or else extending to the end of the file

localization

process of adapting a collection of [resources](#def_resource) to a language, region, or culture

log file

output file produced by a [static analysis tool](#def_static_analysis_tool), which enumerates the [results](#def_result) produced by the tool

(log file) viewer

[SARIF consumer](#def_sarif_consumer) that reads a [log file](#def_log_file), displays a list of the [results](#def_result) it contains, and allows an [end user](#def_end_user) to view each result in the context of the [programming artifact](#def_programming_artifact) in which it occurs

logical location

location specified by reference to a programmatic construct, without specifying the [programming artifact](#def_programming_artifact) within which that construct occurs

Example: A class name, a method name, a namespace.

message string

human-readable string that conveys information relevant to an element in a SARIF file

namespaced tag

[tag](#def_tag) in the format <component>{/<component>}\*, for example, "CWE/22"

nested file

[file](#def_file) which is contained within another file

nested logical location

[logical location](#def_logical_location) that is nested within another logical location

Example: A method within a class in C++

newline sequence

sequence of one or more characters representing the end of a line of text

NOTE: Some systems represent a newline sequence with a single newline character; others represent it as a carriage return character followed by a newline character.

parent (file)

[file](#def_file) which contains one or more nested files

physical location

location specified by reference to a [programming artifact](#def_programming_artifact), possibly together with a [region](#def_region) within that artifact

plain text message

[message string](#def_message_string) which does not contain any formatting information

(programming) artifact

[file](#def_file), produced manually by a person or automatically by a program, which results from the activity of programming

Example: Source code, object code, program configuration data, documentation.

problem

[result](#def_result) which indicates a condition that has the potential to detract from the quality of the program

Example: A security vulnerability, a deviation from contractual or legal requirements, a deviation from stylistic standards.

property bag

JSON object consisting of a set of properties with arbitrary [camelCase names](#def_camelCase_name)

region

contiguous portion of a [file](#def_file)

response file

[file](#def_file) containing arguments for a [tool](#def_static_analysis_tool), which are interpreted as if they had appeared directly on the command line

resource

item that requires [localization](#def_localization), such as a [message string](#def_message_string) or [rule metadata](#def_rule_metadata)

result

condition present in a [programming artifact](#def_programming_artifact) and reported by a [static analysis tool](#def_static_analysis_tool)

result file

[file](#def_file) in which a [static analysis tool](#def_static_analysis_tool) detects a [result](#def_result)

result management system

software system that consumes the [log files](#def_log_file) produced by [static analysis tools](#def_static_analysis_tool), produces reports that enable software development teams to assess the quality of their software [artifacts](#def_programming_artifact) at a point in time and to observe trends in the quality over time, and performs functions such as filing bugs and displaying information about individual [results](#def_result)

NOTE: A result management system can interact with a [log file viewer](#def_log_file_viewer) to display information about individual defects.

result matching procedure

algorithm or heuristic by which a [SARIF consumer](#def_sarif_consumer) determines whether two distinct reports of a [result](#def_result) logically represent the same result

rich text message

[message string](#def_message_string) which contains formatting information such as Markdown formatting characters

rule

specific criterion for correctness verified by a [static analysis tool](#def_static_analysis_tool)

NOTE 1: Many static analysis tools associate a “rule id” with each [result](#def_result) they report, but some do not.

NOTE 2: Some rules verify generally accepted criteria for correctness; others verify conventions in use in a particular team or organization.

Example: “Variables must be initialized before use”, “Class names must begin with an uppercase letter”.

rule configuration information

[rule metadata](#def_rule_metadata) that a [tool](#def_static_analysis_tool) can modify at runtime, before executing its scan

rule id

[stable value](#def_stable_value) which a [static analysis tool](#def_static_analysis_tool) associates with a [rule](#def_rule)

NOTE: A rule id is more likely to remain stable if it is a symbolic or numeric value, as opposed to a descriptive string.

Example: CA2001

rule metadata

information that describes a [rule](#def_rule)  
  
Example: id, description, category, author

run

1. invocation of a specified [static analysis tool](#def_static_analysis_tool) on a specified version of a specified set of [analysis targets](#def_analysis_target), with a specified set of runtime parameters

2. set of [results](#def_result) produced by such an invocation

SARIF consumer

program that reads and interprets a SARIF log file

SARIF log file

[log file](#def_log_file) in the format defined by the SARIF specification

SARIF producer

program that emits output in the SARIF format

SARIF resource file

file containing [resources](#def_resource) for a single language, in the format defined by the SARIF specification

stable value

value which, once established, never changes over time

(static analysis) tool

program that examines [programming artifacts](#def_programming_artifact) to detect problems, without executing the program

Example: Lint

tag

string that conveys additional information about the SARIF [log file](#def_log_file) element to which it applies

taint analysis

the process of tracing the path of tainted data through a program

tainted data

data that enters a program from an untrusted source, such as user input

text file

[file](#def_file) considered as a sequence of characters organized into [lines](#def_line) and [columns](#def_column)

text region

[region](#def_region) representing a contiguous range of zero or more character in a [text file](#def_text_file)

top-level file

[file](#def_file) which is not contained within any other file

Example: Category (for example, “Style” or “Security”), documentation URI.

top-level logical location

[logical location](#def_logical_location) that is not nested within another logical location

Example: A global function in C++

triage

decide whether a [result](#def_result) indicates a [problem](#def_problem) that needs to be corrected

user

see [end user](#def_end_user).

viewer

see [log file viewer](#def_log_file_viewer).

## Normative References

[BCP14] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, March 1997, https://tools.ietf.org/html/bcp14.

[GFM] “GitHub-Flavored Markdown spec”, Version 0.28-gfm (2017-08-01), https://github.github.com/gfm/.

[ISO639-1] “Codes for the representation of names of languages -- Part 1: Alpha-2 code”, ISO 639-1:2002, July 2002, <https://www.iso.org/standard/22109.html>.

[ISO639-2] “Codes for the representation of names of languages -- Part 2: Alpha-3 code”, ISO 639-2:1998, October 1998, <https://www.iso.org/standard/4767.html>.

[ISO639-3] “Codes for the representation of names of languages -- Part 3: Alpha-3 code for comprehensive coverage of languages”, ISO 639-3:2007, February 2007, <https://www.iso.org/standard/39534.html>.

[ISO8601:2004] “Data elements and interchange formats -- Information interchange -- Representation of dates and times”, ISO 8601:2004, December 2004, <https://www.iso.org/standard/40874.html>.

[ISO14977:1996] “Information technology – Syntactic metalanguage – Extended BNF”, ISO/IEC 14977:1996(E), December 1996, <https://www.iso.org/standard/26153.html>.

[JSCHEMA01]Wright, A., “JSON Schema: A Media Type for Describing JSON Documents”, April 2017 (expires October 2017), <http://json-schema.org/latest/json-schema-core.html>.

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

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5646] Phillips, A., Ed., and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <http://www.rfc-editor.org/info/rfc5646>.

[RFC7763] Leonard, S., "The text/markdown Media Type", RFC 7763, DOI 10.17487/RFC7763, March 2016, <http://www.rfc-editor.org/info/rfc7763>.

[RFC7764] Leonard, S., "Guidance on Markdown: Design Philosophies, Stability Strategies, and Select Registrations", RFC 7764, DOI 10.17487/RFC7764, March 2016, <http://www.rfc-editor.org/info/rfc7764>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <http://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259, DOI 10.17487/RFC8259, December 2017, <http://www.rfc-editor.org/info/rfc8259>.

[SEMVER] “Semantic Versioning 2.0.0”, <http://semver.org/>.

[UNICODE10] Unicode 10.0, June 2017, <http://www.unicode.org/versions/Unicode10.0.0/>

## Non-Normative References

[CMARK] “CommonMark Spec”, Version 0.28, (2017-08-01), <http://spec.commonmark.org/0.28/>.

[CWE] “Common Weakness Enumeration”, <https://cwe.mitre.org>.

[GFMCMARK] “GitHub's fork of cmark, a CommonMark parsing and rendering library and program in C”, <https://github.com/github/cmark>.

[GFMENG] “GitHub Engineering: A formal spec for GitHub Flavored Markdown”, <https://githubengineering.com/a-formal-spec-for-github-markdown/>.

[ISO9899:2011] “Information technology – Programming languages – C”, ISO/IEC 9899, December 2011, <https://www.iso.org/standard/57853.html>.

[ISO14882:2014] “Information technology – Programming languages – C++”, ISO/IEC 14882, December 2014, <https://www.iso.org/standard/64029.html>.

[ISO23270:2006] “Information technology – Programming languages – C#”, ISO/IEC 23270, September 2006, <https://www.iso.org/standard/42926.html>.

# Conventions

## General

The following conventions are used within this document.

## Format examples

This document contains several partial examples of the SARIF format. The examples are formatted for clarity, as permitted by [[RFC8259](#RFC8259)], which allows “insignificant whitespace” before or after any token; implementations do not need to follow the whitespace convention used in these examples. In these examples, an ellipsis (…) is used to indicate that portions of the log file text required by this specification have been omitted for brevity. A ‘#’ character introduces a comment that extends to the end of the line. These comments are present for explanatory purposes and are not part of the SARIF file format. When a JSON string is too long to fit on a line, it is broken into multiple lines. This is not part of the SARIF format, since JSON strings cannot contain control characters such as newlines.

## Property notation

A JSON object consists of a set of properties. The value of a property can itself be an object, allowing arbitrary nesting. When necessary for clarity or to avoid ambiguity, we use the “dot” notation to refer to nested values. For example, the physicalLocation object defines a property region whose value is a region object, which in turn contains a length property. For clarity, we can refer to the length property as physicalLocation.region.length.

## Syntax notation

Where this specification describes a syntactic construct, it uses the extended Backus-Naur form (EBNF) defined in [[ISO14977:1996](#ISO14977)].

In all EBNF definitions in this spec:

1. The following syntax rules are assumed:

decimal digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

non negative integer =

"0"

| decimal digit – '0', { decimal digit };

1. The following “special sequence” (see [[ISO14977:1996](#ISO14977)], §4.19 and §5.11) refers to any character that can appear in a JSON string according to [[ECMA404](#ECMA404)]:

? JSON string character ?

# File format

## General

A SARIF log file **SHALL** contain the results of a one or more analysis runs. The runs do not need to be produced by the same analysis tool.

A SARIF log file **SHALL** conform to the requirements of the JSON format. The top-level value in the log file **SHALL** conform to the JSON object grammar; that is, it **SHALL** consist of a comma-separated sequence of name/value pairs, enclosed in curly brackets, as described in [[RFC8259](#RFC8259)]. We refer to the object represented by this top-level value as the sarifLog object (§3.10).

A SARIF log file **SHALL** be encoded in UTF-8 [[RFC3629](#RFC3629)].

NOTE: [[RFC8259](#RFC8259)] requires this encoding for any JSON text “exchanged between systems that are not part of a closed ecosystem.”

## fileContent objects

### General

Certain properties in this specification represent the contents of portions of external files, for example, files that were scanned by an analysis tool. SARIF represents such file content with a fileContent object. Depending on the circumstances, the SARIF log file might need to represent this content as readable text, raw bytes, or both.

### text property

If the external file is a text file, a fileContent object **SHOULD** contain a property named text whose value is a string containing the relevant text. Since SARIF log files are encoded in UTF-8 ([[RFC3629](#RFC3629)]; see §3.1), this means that if the external file is a text file in any encoding other than UTF-8, the SARIF producer **SHALL** transcode the text to UTF-8 before assigning it to the text property. The SARIF producer **SHALL** escape any characters that [[RFC8259](#RFC8259)] requires to be escaped.

If the external file is a binary file, the text property **SHALL** be absent.

### binary property

If the external file is a binary file, or if the SARIF producer cannot determine whether the external file is a text file or a binary file, a fileContent object **SHALL** contain a property named binary whose value is a string containing the MIME Base64 encoding [[RFC2045](#RFC2045)] of the bytes in the relevant portion of the file.

If the external file is a text file in an encoding other than UTF-8, the binary property **MAY** be present, in which case it **SHALL** contain the MIME Base64 encoding of the bytes representing the relevant text in its original encoding.

If the external file is a UTF-8 text file, the binary property **SHOULD** be absent. If it is present, it **SHALL** contain the MIME Base64 encoding of the UTF-8 bytes representing the relevant text.

## fileLocation objects

### General

Certain properties in this specification specify the location of a file. SARIF represents a file location with a fileLocation object. The most important member of a fileLocation object is its uri property (§3.3.2). If the value of the uri property is a relative URI, the uriBaseId property (§3.3.3) can sometimes be used to resolve the relative URI to an absolute location.

### uri property

A fileLocation object **SHALL** contain a property named uri whose value is a string containing a valid URI as described in [[RFC3986](#RFC3986)].

If the URI refers to a file stored in a version control system (VCS), the value **SHALL** preserve relevant details that permit the target file to be retrieved from the VCS. If the URI refers to a file stored on a physical file system, it **MAY** be specified as a relative URI that omits root information details (such as hard drive letter and an arbitrarily named root directory associated with a source code enlistment).

NOTE 1: An absolute URI might contain information that represents unwanted information disclosure, particularly in cases where a tool is analyzing files stored on a physical file system. For example, a file path might contain the account name of a developer.

If the file is a nested file, then the property name **SHALL** be the URI of the outermost parent, together with a fragment that describes the nesting of the file within its parent or parents. The fragment **SHALL** begin with a forward slash character ("/") to emphasize that it represents the complete path to the nested file within its container. This requirement allows SARIF consumers to look up the URI of a nested file in the dictionary contained in the run.files property (§3.11.11).

Two URIs **SHALL** be considered equivalent if their normalized forms are the same, as described in [[RFC3986](#RFC3986)].

NOTE 2: For example, in the normalized form specified in RFC 3986:

* Percent-encoded characters use upper-case hexadecimal digits.
* Characters in the ALPHA and DIGIT ranges are not be percent-encoded, nor are hyphen, underscore, or tilde.
* The “:” delimiter is omitted if the port component of the authority is empty.
* In the host component, registered names and hexadecimal addresses use lower-case.

When two URIs are not equivalent in this sense (that is, when their normalized forms are not the same), we will say that they are “distinct.”

Aside from normalization, SARIF producers **SHALL NOT** make any other changes to the text of the URI; for example, they **SHALL NOT** convert the URI path to upper case or to lower case.

NOTE 3: This is especially important when the same SARIF file might be consumed on multiple platforms, for example, a platform such as Windows, whose NTFS file system is case-insensitive but case-preserving, and a platform such as Linux, whose file system is case-sensitive. Consider a scenario where a tool runs on a Windows system using NTFS, and the tool decides to lower-case the file names in the log. If the source files and the SARIF log were transferred to a Linux system, the URIs in the log file would not match the path names on the destination system.

### uriBaseId property

If the value of its uri property (§3.3.2) is a relative URI, a fileLocation object **MAY** contain a property named uriBaseId whose value is a string which indirectly specifies the absolute URI with respect to which that relative URI is interpreted. If the uri property contains an absolute URI, the uriBaseId property **SHALL** be absent.

To avoid ambiguity in interpreting the property names (§3.11.11.2) in the run.files object (§3.11.11), the value of the uriBaseId property **SHALL NOT** include the character "#".

If a SARIF consumer requires an absolute URI (for example, to display the specified file to a user), then it needs to have the necessary information to resolve the value of the uriBaseId property to an absolute URI, which can then be combined with the relative URI stored in the uri property. One possibility is for the SARIF producer and consumers to agree on the meanings of any values for the uriBaseId property that appear in the log file. Another possibility is for the end user to supply those meanings to the consumer, either on the consumer’s command line, or through a user interface prompt.

EXAMPLE 1: In this example the SARIF consumer’s command line specifies that any uriBaseId property whose value is "SRCROOT" refers to the absolute URI "file:///C:/browser/src":

C:> SarifAnalyzer --input log.sarif --uriBaseId SRCROOT="file:///C:/browser/src"

The value of the uriBaseId property can be any string; it does not need to have any particular syntax or follow any particular naming convention. In particular, it does not need to designate a machine environment variable or similar value, although it might. The SARIF producer and any SARIF consumers need to agree on the meanings of any values for the uriBaseId property that appear in the log file.

EXAMPLE 2: In this example, the analysis tool has set the uri property of resultFile.fileLocation (§3.21.3) to the relative URI of the file in which a result was detected. The tool has also set the value of the uriBaseId property to "%srcroot%". The analysis tool and the SARIF consumers have agreed upon a convention whereby this indicates that the relative URI is expressed relative to the root of the source tree in which the file appears.

"resultFile": {

"fileLocation": {

"uri": "drivers/video/hidef/driver.c",

"uriBaseId": "%srcroot%"

}

}

EXAMPLE 3: In this example, the analysis tool has set the uri property of analysisTarget.fileLocation to the relative URI of the file which the tool was instructed to scan. The tool has also set the value of the uriBaseId property to "$bindrop". The analysis tool and the SARIF consumers have agreed upon a convention whereby this indicates that the relative URI is expressed relative to the directory containing the binary files produced by a build.

"analysisTarget": {

"fileLocation": {

"uri": "hidef.dll",

"uriBaseId": "$bindrop"

}

}

NOTE: There are various reasons for providing the uriBaseId property:

* Portability: A log file that contains relative URIs together with uriBaseId properties can be interpreted on a machine where the files are located at a different absolute location.
* Determinism: A log file that uses uriBaseId properties has a better chance of being “deterministic”; that is, of being identical from run to run if none of its inputs have changed, even if those runs occur on machines where the files are located at different absolute locations. For more information on this point, see Appendix F, “Producing deterministic SARIF log files”.
* Security: The use of uriBaseId properties avoids the persistence of absolute path names in the log file. Absolute path names can reveal information that might be sensitive.
* Semantics: Assuming the reader of the log file (an end user or another tool) has the necessary context, they can understand the meaning of the location specified by the uri property, for example, “this is a source file”.
* Brevity: The uriBaseId property might be shorter than the absolute path it represents.

## String properties

Unless otherwise specified in the description of a specific property, all properties whose values are of type "string" **SHALL** have a non-empty value.

## Object properties

Certain properties in this specification are defined to be JSON objects whose property names satisfy certain conditions. Examples are the run.files property (§3.11.11) and the rule.messageStrings property (§3.30.7). Unless otherwise specified in the description of a specific property, if any such object is empty, then either the property **SHALL** be represented as an empty object {}, or it **SHALL** be absent.

## Array properties

### General

Certain properties in this specification are defined to be JSON arrays. Examples are the invocation.toolNotifications property (§3.13.20) and the file.hashes property (§3.16.7). Unless otherwise specified in the description of a specific property, if any such array is empty, then either the property **SHALL** be represented as an empty array [], or it **SHALL** be absent.

### Array properties with unique values

Certain array-valued properties in this specification are described as having “unique” elements. When a property is so described, it means that no two elements of the array **SHALL** have equal values. For purposes of this specification, two array elements **SHALL** be considered equal when they satisfy the condition for equality described in [[JSCHEMA01](#JSCHEMA01)], §4.3, “Instance equality”.

## Property bags

### General

Certain properties in this specification are defined to be “property bags”. A property bag is a JSON object (§3.5) containing an arbitrary set of properties. The names of the properties **SHOULD** be camelCase strings, but see [Appendix D](#AppendixConverters) for exceptions. The values of the properties **MAY** be of any JSON type, including strings, numbers, arrays, objects, Booleans, and null. If the value of a property is a string, it **MAY** be an empty string.

### Tags

#### General

If a property bag contains a property with the name tags, then the value of that property **SHALL** be an array containing zero or more arbitrary strings, no two of which **SHALL** be the same. Two strings **SHALL** be considered the same if they consist of the same sequence of Unicode [[UNICODE10](#UNICODE10)] code points.

#### Namespaced tags

Any string in the tags array of any SARIF element **MAY** consist of a forward-slash-separated sequence of components, as follows:

tag = component, { "/", component };

component = component character, { component character };

component character = ? JSON string character ? - "/";

EXAMPLE: "CWE/22"

A tag in this format is referred to as a namespaced tag.

NOTE: SARIF producers can use this feature, for example, to categorize scan results according to a taxonomy such as the Common Weakness Enumeration [[CWE](#CWE)].

#### Tag metadata

A SARIF log file **MAY** provide additional information about any tag value by including a property whose name is the same as that tag value, and whose value is any JSON value. If present, this property **SHALL** be located either in the same property bag that contains the tag, or in the property bag of any SARIF element which lexically contains the element containing the tag.

EXAMPLE: Suppose a SARIF-producing tool classifies results according to the Common Weakness Enumeration, using a tool-specific convention that the tag "CWE/*n*" denotes a result to which CWE *n* applies. Suppose this tool produces the following result:

{ # A result object (§3.18)

"ruleId": "SEC0251",

"message": {

"text": "The path 'data/../bin' is not within the 'data' directory"

},

"properties": {

"tags": [

"security",

"CWE/22"

]

}

}

Now suppose the tool wishes to provide additional information about CWE 22. It might provide that information within the property bag containing the tag (that is, in this example, the property bag belonging to the result object):

{ # A result object (§3.18)

"ruleId": "SEC0251",

"message": {

"text": "The path 'data/../bin' is not within the 'data' directory"

},

"properties": {

"tags": [

"security",

"CWE/22"

],

"CWE/22": {

"description": "Improper Limitation of a Pathname",

"url": "https://cwe.mitre.org/data/definitions/22.html"

}

}

}

However, there might be several results associated with CWE 22. To avoid duplicating the metadata, the tool might choose to place it in the property bag belonging to the run object (§3.11) that lexically contains the result object:

{ # A run object (see §3.11)

"results": [

{

"ruleId": "SEC0251",

"message": {

"text": "The path 'data/../bin' is not within the 'data' directory"

},

"properties": {

"tags": [

"security",

"CWE/22"

]

}

}

],

"properties": { # The run object's property bag.

"CWE/22": {

"description": "Improper Limitation of a Pathname",

"url": "https://cwe.mitre.org/data/definitions/22.html"

}

}

}

## Date/time properties

Certain properties in this specification specify a date and time. The value of every such property, if present, **SHALL** be a string in the following format, which is compatible with [[ISO8601:2004](#ISO86012004)]:

date time = date, "T", time, "Z" (\* UTC time \*);

date = year, "-", month, "-", day;

year = 4 \* decimal digit;

month = 2 \* decimal digit (\* from 01 to 12 \*);

day = 2 \* decimal digit (\* from 01 to 31 \*);

time = hour, ":", minute, ":", second, [ ".", fraction ];

hour = 2 \* decimal digit (\* from 00 to 12 \*);

minute = 2 \* decimal digit (\* from 00 to 59 \*);

second = 2 \* decimal digit (\* from 00 to 60, to accommodate leap second \*);

fraction = decimal digit, { decimal digit };

EXAMPLES:  
2016-02-08T16:08:25Z  
2016-02-08T16:08:25.943Z

A SARIF producer **SHOULD** base the number of digits in fraction on the precision of the clock on the computer on which it runs.

## message objects

### General

Certain objects in this specification define messages intended to be viewed by a user. SARIF represents such a message with a message object, which offers the following features:

* Message strings in plain text (“plain text messages”).
* Message strings that incorporate formatting information (“rich text messages”).
* Message strings with placeholders for variable information.
* Localized message strings.

### Plain text messages

A plain text message **SHOULD** be expressed as a single paragraph of plain text, consisting of one or more complete sentences, each ending with a period (or appropriate punctuation for the language in which the message is written). The message **SHALL NOT** contain formatting information such as HTML tags. The message **SHOULD NOT** contain JSON escaped line breaks (\r or \n). The message string **MAY** contain placeholders (§3.9.4) and embedded links (§3.9.5).

If the message consists of more than one sentence, its first sentence **SHOULD** provide a useful summary of the message, suitable for display in cases where UI space is limited.

NOTE 1: If a tool does not construct the message in this way, the initial portion of the message that a viewer displays where UI space is limited might not be understandable.

NOTE 2: The rationale for these guidelines is that the SARIF format is intended to make it feasible to merge the outputs of multiple tools into a single user experience. A uniform approach to message authoring enhances the quality of that experience.

### Rich text messages

#### General

Rich text messages **MAY** be of arbitrary length and **SHOULD** contain formatting information. The message string **MAY** also contain placeholders (§3.9.4) and embedded links (§3.9.5).

Every rich text message in a given run **SHALL** be expressed in the same markup language, specified by the run.richMessageMimeType property (§3.11.15). For maximum interoperability among SARIF log files produced by different tools, direct producers **SHALL** express rich text messages in GitHub-Flavored Markdown [[GFM](#GFM)]. Since GFM is a superset of CommonMark [[CMARK](#CMARK)], any CommonMark Markdown syntax is acceptable.

If an analysis tool produces a custom output format that includes rich text messages in a format other than GFM, a converter which translates the output of that tool to SARIF **SHOULD NOT** attempt to translate the messages to GFM. Instead, it **SHOULD** set run.richMessageMimeType to a value appropriate to the analysis tool’s output format.

#### Security implications

If the rich text message format is any variant of Markdown, then for security reasons, SARIF producers and SARIF consumers **SHALL** adhere to the following:

* SARIF producers **SHALL NOT** emit messages that contain HTML, even though all variants of Markdown permit it.
* Deeply nested markup can cause a stack overflow in the Markdown processor [[GFMENG](#GFMENG)]. To reduce this risk, SARIF consumers **SHALL** use a Markdown processor that is hardened against such attacks. One example is the GitHub fork of the cmark Markdown processor [[GFMCMARK](#GFMCMARK)].
* To reduce the risk posed by possibly malicious SARIF files that do contain arbitrary HTML (including, for example, javascript: links), SARIF consumers **SHALL** either disable HTML processing (for example, by using an option such as the --safe option in the cmark Markdown processor) or run the resulting HTML through an HTML sanitizer.

SARIF consumers that are not prepared to deal with the security implications of rich text messages **SHALL NOT** attempt to render them and **SHALL** instead fall back to the corresponding plain text messages.

### Messages with placeholders

A message string **MAY** include or more “placeholders.” The syntax of a placeholder is:

placeholder = "{", index, "}"

index = non negative integer

index represents a 0-based index into the array of strings contained in the arguments property (§3.9.11).

When a SARIF consumer displays the message, it **SHALL** replace every occurrence of the placeholder {n} with the string value at index n in the arguments array. Within both plain text and rich text message strings, the characters “{” and “}” **SHALL** be represented by the character sequences “{{” and “}}” respectively.

Within a given message object:

* The plain text and rich text message strings **MAY** contain different numbers of placeholders.
* A given placeholder index **SHALL** have the same meaning across all the message strings in the object (so that they can be replaced with the same element of the arguments array).

EXAMPLE: Suppose a message object’s text property (§3.9.7) contains this string:

"The variable \"{0}\" defined on line {1} is never used. Consider removing \"{0}\"."

There are two distinct placeholders, {0} and {1} (although {0} occurs twice). Therefore, the arguments array will have at least two elements, the first corresponding to {0} and the second corresponding to {1}.

### Messages with embedded links

A message string **MAY** include one or more links to locations within files mentioned in the enclosing result object (§3.18). We refer to these links as “embedded links”.

The syntax of an embedded link is:

embedded link = "[", link text, "](", link target, ")";

link text = ? JSON string character ? – "]"

link target = non negative integer;

link text is the message text visible to the user. If the link occurs within a plain text message (§3.9.2), link text **SHALL** be plain text. If the link occurs within a rich text message (§3.9.3), link text **MAY** be either plain text or rich text.

Square brackets (("[" and "]") in both plain text messages and rich text messages **SHALL** be escaped with a backslash ("\") to prevent them from being interpreted as embedded links. Since JSON itself treats the backslash as an escape character, the backslash**SHALL** be doubled.

EXAMPLE 1:

"message": {

"text": "This is not part of an embedded link: \"\\[\"." # See §3.9.7

}

The result object (§3.18) within which the message object occurs **SHALL** contain exactly one physicalLocation object (§3.21) whose id property (§3.21.2) is equal to the value of link target.

NOTE: link target is required to be an integer, rather than arbitrary string, to avoid confusion with normal Markdown link syntax. Negative values are forbidden because their use would suggest some non-obvious semantic difference between positive and negative values.

EXAMPLE 2: In this example, a plain text message contains an embedded link to a location with a file. There is exactly one physicalLocation object whose id property matches the link target.

{

"version": "2.0.0",

"runs": [

{

"results": [

{

"ruleId": "TNT0001",

"message": {

"text": "Tainted data was used. The data came from [here](3)."

},

"locations": [

{

"analysisTarget": {

"uri": "file:///C:/code/main.c",

"region": {

"startLine": 15,

"startColumn": 9

}

}

}

],

"relatedLocations": [

{

"physicalLocation": {

"id": 3,

"uri": "file:///C:/code/input.c",

"region": {

"startLine": 15,

"startColumn": 9

}

}

}

]

}

]

}

]

}

### Message string resources

#### General

A message object can directly contain message strings in its text (§3.9.7) and richText (§3.9.8) properties. It can also indirectly refer to message strings through its messageId (§3.9.9) and richMessageId (§3.9.10) properties. We refer to these indirectly referenced message strings as “message string resources,” and we refer to the contents of the messageId and richMessageId properties as “resource identifiers.”

The resource identifiers used for the values of messageId and richMessageId properties **SHALL** be distinct. That is, any given resource identifier **SHALL NOT** appear both as the value of a messageId property and the value of a richMessageId property in the same run.

Resources enable message strings to be localized into other languages. A SARIF run object (§3.11) can optionally contain the message string resources for a single language, namely the language designated by its tool.language property (§3.12.7). We refer to these message strings as “embedded resources.” Embedded message string resources are stored in the run.resources.messageStrings property (§3.29.2).

If a SARIF consumer needs to access resources for a language other than the one specified by tool.language, it can attempt to locate the resources in an external file. We refer to such a file as a “SARIF resource file”, and we refer to the message strings in such a file as “external resources.” §3.9.6.3 defines the naming convention and file lookup procedure for SARIF resources files. §3.9.6.4 defines the SARIF resource file format.

#### Embedded string resource lookup procedure

When a SARIF consumer needs to locate a message string for the run’s declared language, it **SHALL** follow the string lookup procedure specified in this section. The run object **SHALL** contain enough information for the string lookup procedure to succeed. This ensures that a SARIF consumer can always locate the message strings for the declared language without having to consult a SARIF resource file, which might not be available. The string lookup procedure depends on whether the consumer can render rich text messages.

If the consumer can render rich text messages, the string lookup procedure is:

1. If message.richText is present, use its value.
2. Otherwise, if message.richMessageId is present, and run.resources.messageStrings is present and contains a property whose name matches message.richMessageId, use the value of that property.
3. Otherwise, execute the lookup procedure for plain text messages, below.

If the consumer cannot render rich text messages, the string lookup procedure is:

1. If message.text is present, use its value.
2. Otherwise, if message.messageId is present, and run.resources.messageStrings is present and contains a property whose name matches message.messageId, use the value of that property.
3. Otherwise, the string lookup procedure fails (which means that the SARIF log file is invalid).

#### SARIF resource file lookup procedure

When a SARIF consumer needs to locate a message string for a language other than the tool’s declared language, it **SHALL** follow the file lookup procedure specified in this section to locate a SARIF resource file.

SARIF resource file names **SHALL** follow the naming convention defined by the following syntax:

SARIF resource file name = language tag, ".resources.sarif"

language tag = ? RFC 5646 language tag ?

The file lookup procedure is:

1. Determine the “resource URI base” as follows:  
   1. If the SARIF consumer is configured to obtain resources from a particular location (for example, by means of a configuration file or a command line argument), that is the resource URI base.
   2. If the resource URI base has not yet been determined, and if run.tool.resourceLocation (§3.12.8) is present:  
      1. If run.tool.resourceLocation.uri is an absolute URI, that is the resource URI base.
      2. If the resource URI base has not yet been determined, then if run.tool.resourceLocation.uriBaseId is present and run.originalUriBaseIds is present and contains a matching property, then the resource URI base is the absolute URI obtained by combining run.tool.resourceLocation.uri with the matching property value from run.originalUriBaseIds
   3. If the resource URI base has not yet been determined, the SARIF consumer **MAY** use other means to determine it. (For example, it might prompt the user).
   4. If the resource URI base has not yet been determined, the file lookup procedure fails.
2. Locate a SARIF resource file under the resource URI base location as follows:  
   1. Construct a file name using the full RFC 5646 language tag specified by the user. (For example, this might be the operating system’s current UI language, such as fr-FR. In this case, the file name would be fr-FR.resources.sarif.) If a file by that name is present, use it.
   2. Otherwise, if the first subtag is one of the two- or three-letter primary language subtags defined in [[ISO639-1](#ISO639_1)], [[ISO639-2](#ISO639_2)] or, [[ISO639-3](#ISO639_3)], construct a file name using only that subtag. (Continuing the previous example, the file name would be fr.resources.sarif.) If a file by that name is present, use it.
   3. If the SARIF resource file name has not yet been determined, the SARIF consumer **MAY** use other means to determine it. (For example, it might prompt the user.)
   4. If the SARIF resource file name has not yet been determined, the file lookup procedure fails.

If the file lookup procedure fails, the SARIF consumer **MAY** follow the string lookup procedure for embedded resources specified in §3.9.6.2. In that case, the SARIF consumer might display messages in a language other than the one the end user requested. The SARIF consumer **MAY** notify the user if it was unable to locate resources for the requested language.

If the file lookup procedure succeeds, the SARIF consumer **SHALL** follow the string lookup procedure defined in §3.9.6.2 to extract the required message string from the SARIF resource file.

#### SARIF resource file format

##### General

A SARIF resource file contains only that subset of the elements of a SARIF log file that are necessary to describe resources. Some of the elements that are present in a SARIF resource file are constrained differently than they are in a SARIF log file, for example, by being required rather than optional, or by having a different number of array elements. All these differences are described in the sections that follow.

##### sarifLog object

The root element of a SARIF resource file is a sarifLog object (§3.10). Its permitted properties, and their differences from the corresponding elements in a SARIF log file, are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **Required?** | **Difference from SARIF log file** |
| $schema (§3.10.3) | string | No | Specifies the URI from which the JSON schema for the SARIF resource file format (rather than the SARIF log file format) can be obtained. |
| runs (§3.10.4) | run[] (§3.11) | Yes | Array contains exactly one element, rather than one or more. That element contains only the properties specified in §3.9.6.4.3. |

##### run object

The permitted properties on the run object, and their differences from the corresponding elements in a SARIF log file, are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **Required?** | **Difference from SARIF log file** |
| tool (§3.11.7) | tool (§3.12) | Yes | Required rather than optional. Contains only the properties specified in §3.9.6.4.4. |
| resources (§3.11.14) | resources (§3.29) | Yes | Required rather than optional. |

##### tool object

The permitted properties on the tool object, and their differences from the corresponding elements in a SARIF log file, are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **Required?** | **Difference from SARIF log file** |
| name (§3.12.2) | string | Yes | None |
| fullName (§3.12.3) | string | No | None |
| semanticVersion (§3.12.4) | string | Yes | None |
| version (§3.12.5) | string | No | None |
| fileVersion (§3.12.6) | string | No | None |
| language (§3.12.7) | string | Yes | Required rather than recommended. Just as in a SARIF log file, it specifies the language of the resources embedded in the file. |

##### resources object

The resources object in a SARIF resource file is identical to the resources object in a SARIF log file (§3.29).

### text property

A message object **MAY** contain a property named text whose value is a non-empty string containing a plain text message (§3.9.2).

### richText property

A message object **MAY** contain a property named richText whose value is a non-empty string containing a rich text message (§3.9.3).

If the richText property is present, the text property (§3.9.7) **SHALL** also be present. This ensures that the message is viewable even in contexts that do not support the rendering of rich text.

SARIF consumers that cannot (or choose not to) render rich text **SHALL** ignore the richText property and use the text property instead.

### messageId property

A message object **MAY** contain a property named messageId whose value is a non-empty string containing the resource identifier (§3.9.6) for the desired plain text message (§3.9.2). See §3.9.6.2 and §3.9.6.3 for details of the resource string lookup procedure.

### richMessageId property

A message object **MAY** contain a property named richMessageId whose value is a non-empty string containing the resource identifier (§3.9.6) for the desired rich text message (§3.9.3).

SARIF consumers that cannot (or choose not to) render rich text **SHALL** ignore the richMessageId property and use the messageId property instead. See §3.9.6.2 and §3.9.6.3 for details of the resource string lookup procedure.

### arguments property

If the message string specified by any of the properties text (§3.9.7), richText (§3.9.8), messageId (§3.9.9), or richMessageId (§3.9.10) contains any placeholders (§3.9.4), the message object **SHALL** contain a property named arguments whose value is an array of strings. §3.9.4 specifies how a SARIF consumer combines the contents of the arguments array with the message string to construct the message that it presents to the end user, and provides an example.

If none of the properties text, richText, messageId, or richMessageId contains any placeholders, the arguments property **SHALL** be absent.

The arguments array **SHALL** contain as many elements as required by the maximum placeholder index among all the message strings specified by the text, richText, messageId, or richMessageId properties.

EXAMPLE: If the highest numbered placeholder in the text message string is {3} and the highest numbered placeholder in the richText message string is {5}, the arguments array must contain at least 6 elements.

## sarifLog object

### General

A sarifLog object specifies the version of the file format and contains the output from one or more runs.

EXAMPLE:

{

"version": "2.0.0", # See §3.10.2.

"runs": [ # See §3.10.4.

{

... # A run object (§3.11)

},

...

{

... # Another run object

}

]

}

### version property

A sarifLog object **SHALL** contain a property named version whose value is a string designating the version of the SARIF format to which this log file conforms. This string **SHALL** have the value "2.0.0".

Although the order in which properties appear in a JSON object value is not semantically significant, the version property **SHOULD** appear first.

NOTE: This will make it easier for parsers to handle multiple versions of the SARIF format, if new versions are defined in the future.

### $schema property

A sarifLog object **MAY** contain a property named $schema whose value is a string containing a URI from which a JSON schema document describing the version of the SARIF format to which this log file conforms can be obtained.

If the $schema property is present, the JSON schema obtained from the specified URI **SHALL** describe the version of the SARIF format specified by the version property (§3.10.2).

NOTE: The purpose of the $schema property is to allow JSON schema validation tools to locate an appropriate schema against which to validate the log file. This is useful, for example, for tool authors who wish to ensure that logs produced by their tools conform to the SARIF format.

### runs property

A sarifLog object **SHALL** contain a property named runs whose value is an array of one or more run objects (§3.11).

## run object

### General

A run object describes a single run of an analysis tool and contains the output of that run.

EXAMPLE:

{

"tool": { # See §3.11.7.

... # A tool object (§3.12).

},

"results": [ # See §3.11.13.

{

... # A result object (§3.18).

},

...

{

... # Another result object.

}

]

}

### id property

A run object **MAY** contain a property named id whose value is a string which uniquely identifies the run.

NOTE: A result management system can use run.id to associate the information in the log with additional information not provided by the analysis tool that produced it.

### stableId property

A run object **MAY** contain a property named stableId whose value is a string containing a stable identifier for the run. Multiple runs of the same type **MAY** have the same stableId.

EXAMPLE:

{

"stableId": "Nightly security scanner run"

}

### baselineId property

A run object **MAY** contain a property named baselineId whose value is a string which **SHALL** match the id property (§3.11.2) of some previous run.

If the baselineId property is present, the result.baselineState property (§3.18.16) of every result object (§3.18) in the current run **SHALL** be computed with respect to the run specified by baselineId.

If the baselineId property is absent, and any result object has a value for its baselineState property, there needs to be out of band information available to determine the run with respect to which result.baselineState was computed.

### automationId property

A run object **MAY** contain a property named automationId whose value is a string containing an identifier that allows the run to be correlated with other artifacts produced by a larger automation process.

EXAMPLE: In an environment where an analysis tool is executed as part of an automated build process, the “build id” assigned by the build system might serve as the automationId, allowing the tool run to be associated with other artifacts produced by the build.

{

...

"runs": [

{

"automationId": "Build-14.0.1.2-20160518-15:48:02",

...

}

]

}

### architecture property

A run object **MAY** contain a property named architecture whose value is a string that specifies the hardware architecture at which the analysis targets are targeted. This does not need to be the same as the architecture on which the analysis tool is executed.

This specification does not specify a set of valid values for the architecture property.

EXAMPLE: An analysis tool running on a x86 architecture might be run once for a set of binaries that target x86, and then again for another set of binaries that target AMD64. The tool might set the architecture property for the first run to "x86", and for the second run to "AMD64".

### tool property

A run object **SHALL** contain a property named tool whose value is a tool object (§3.12) that describes the analysis tool that was run.

### invocations property

A run object **MAY** contain a property named invocations whose value is an array of unique (§3.6.2) invocation objects (§3.13) that describe the invocation of the analysis tool that was run.

Normally, an analysis tool runs as a single process, and the invocations array requires only one element. The invocations property is defined as an array, rather than as a single invocation object, to accommodate tools which execute a sequence of programs to produce results. For example, a tool might run one program to determine the set of files to analyze and another program to analyze those files.

The elements of the invocations array **SHOULD**, as far as possible, be arranged in chronological order according to the start time of each process. If some of the processes run in parallel, this might not be possible.

### conversion property

If a run object was produced by a converter, it **MAY** contain a property named conversion whose value is a conversion object (§3.14) that describes how the converter transformed the analysis tool’s native output format into the SARIF format.

A direct producer **SHALL NOT** emit the conversion property.

### originalUriBaseIds property

A run object **MAY** contain a property named originalUriBaseIds whose value is an object, each of whose property names designates a URI base id (§3.3.3). The value of each property is a valid absolute URI (§3.3.2) which is the value of that URI base id on the machine where the SARIF producer ran.

This property allows SARIF consumers to resolve any relative URIs which appear in any fileLocation objects (§3.2) in the run, as long as the consumer runs either on the same machine as the producer, or on a machine with an identical file system layout. This is useful for individual developers who wish to run analysis tools and examine the results in a viewer. It is also useful for teams which share a convention for their file system layout.

When a SARIF consumer resolves a relative URI in a SARIF file, if the user has configured the consumer to use a particular value for the URI base id, the consumer **SHALL** use the configured value. If the file does not exist in that location, then the consumer **SHALL** use the value specified in the originalBaseIds property, if present. If the file does not exist at that location, the consumer **MAY** use other information or heuristics to locate the file.

EXAMPLE: In this example, the value of the URI base id "SRCROOT" on the machine where the SARIF producer ran was "file:///C:/src". The producer detected a result in a file whose location relative to that URI base id was "lib/memory.c". A viewer which wished to display that file would first attempt to locate it on the local file system at "C:\src\lib\memory.c". If the file did not exist at that location, the viewer might prompt the user for the location.

{ # A run object

"originalBaseIds": {

"SRCROOT": "file:///C:/src"

},

"results": [

{ # A result object (§3.18)

"ruleId": "CA1001",

"locations": [

{ # A location object (§3.20)

"analysisTarget": {

"fileLocation": { # A fileLocation object (§3.2)

"uri": "lib/memory.c",

"uriBaseId": "SRCROOT"

}

}

}

]

}

]

}

### files property

#### General

A run object **SHOULD** contain a property named files whose value is a JSON object (§3.5), each of whose properties represents a file relevant to the run.

The object specified by the files property **SHOULD** contain properties representing at least those files in which results were detected, but it **MAY** contain properties representing all files examined by the tool (whether or not results were detected in those files), or any subset of those files. It **MAY** also include other files relevant to the run, such as attachments (§3.13.5, §3.18.17).

NOTE: file objects contain information that is useful for viewers. Viewers will be able to provide the most information to users if the files property is present and contains information for every file in which results were detected.

EXAMPLE:

"files": {

"file:///C:/Code/main.c": {

"mimeType": "text/x-c",

"hashes": [

{

"value": "b13ce2678a8807ba0765ab94a0ecd394f869bc81",

"algorithm": "sha256"

}

]

}

}

#### Property names

The property names in the files object are related to the file locations specified in fileLocation objects (§3.2) within the run. The syntax for the property names is:

files object property name = absolute property name | relative property name

absolute property name = URI

relative property name = [ uri base id prefix ], relative-ref

URI = (? an absolute URI as defined by the URI construct in RFC 3986 ?)

relative-ref = (? a relative URI as defined by the relative-ref construct in RFC 3986 ?)

uri base id prefix = "#", uri base id, "#"

uri base id = (? the value of a uriBaseId property in a fileLocation object ?)

If the fileLocation.uri property (§3.3.2) contains an absolute URI, the corresponding property name in the files object **SHALL** be an absolute property name containing an absolute URI equivalent to the value of fileLocation.uri in the sense described in §3.3.2.

EXAMPLE 1: In this example, a fileLocation object in the run has a uri property whose value is an absolute URI. The name of the corresponding property in the files object matches that URI.

{ # A run object (§3.11).

"results": [

{ # A result object (§3.18).

"relatedLocations": [

{ # An annocatedCodeLocation object (§3.27).

"physicalLocation": { # A physicalLocation object (§3.21).

"fileLocation": { # A fileLocation object (§3.2).

"uri": "file:///C:/source/input.c"

}

}

}

]

}

],

"files": {

"file:///C:/source/input.c": { # Property name matches absolute URI from

... # fileLocation object

}

}

}

If the fileLocation.uri property contains a relative URI, the corresponding property name in the files object **SHALL** be a relative property name whose relative-ref portion is a relative URI equivalent to the value of fileLocation.uri in the sense described in §3.3.2.

EXAMPLE 2: In this example, a fileLocation object in the run has a uri property whose value is a relative URI. The name of the corresponding property in the files object matches that URI.

{ # A run object (§3.11).

"results": [

{ # A result object (§3.18).

"relatedLocations": [

{ # An annocatedCodeLocation object (§3.27).

"physicalLocation": { # A physicalLocation object (§3.21).

"fileLocation": { # A fileLocation object (§3.2).

"uri": "input.c",

"uriBaseId": "SRCROOT"

}

}

}

]

}

],

"files": {

"input.c": { # Property name matches relative URI from

... # fileLocation object

}

}

}

If two or more properties in the files object correspond to fileLocation objects with equivalent relative URI-valued uri properties but different uriBaseId properties (§3.3.3), then each of the conflicting property names **SHALL** have a uri base id prefix. This avoids a situation where two properties would otherwise have the same property name.

NOTE 1: Since no valid URI starts with a "#" character, there is no danger of a property name that starts with a uri base id prefix colliding with another property name that represents a relative URI with no prefix.

EXAMPLE 3: In this example, two fileLocation objects have the same relative URI-valued uri property but different uriBaseId properties. The names of the corresponding properties in the files object include a uri base id prefix to avoid a property name collision.

{ # A run object (§3.11).

"results": [

{ # A result object (§3.18).

"relatedLocations": [

{ # An annocatedCodeLocation object (§3.27).

"physicalLocation": { # A physicalLocation object (§3.21).

"fileLocation": { # A fileLocation object (§3.2).

"uri": "utilities.c",

"uriBaseId": "SRCROOT"

}

},

"physicalLocation": {

"fileLocation": {

"uri": "utilities.c",

"uriBaseId": "TESTSRCROOT"

}

}

}

]

}

],

"files": {

"#SRCROOT#utilities.c": { # Property name includes uri base id prefix

...

},

"#TESTSRCROOT#utilities.c": {

...

}

}

}

If a relative property name does *not* conflict with any other property name in the files object, the uri base id prefix portion of the property name **SHOULD** be absent (see EXAMPLE 2).

NOTE 2: This recommendation improves the readability of the SARIF log file. It is a recommendation, rather than a requirement, to accommodate SARIF producers which do not wish to include the extra logic necessary to keep track of property name collisions.

Regardless of whether the property name represents an absolute URI, a relative URI, or a relative URI with a uri base id prefix, the URI portion of the property name **SHOULD** be normalized as described in [[RFC3986](#RFC3986)].

EXAMPLE 4: In this example, the uri property of the fileLocation object is not normalized, but the name of the corresponding property in the files object *is* normalized.

{ # A run object

"results": [

{ # A result object

"relatedLocations": [

{ # An annocatedCodeLocation object

"physicalLocation": { # A physicalLocation object

"fileLocation": { # A fileLocation object

"uri": "FILE:///C:/source/input.c" # scheme is not normalized

}

}

}

]

}

],

"files": {

"file:///C:/source/input.c": { # Property name matches absolute URI after

... # normalization (scheme has been normalized).

}

}

}

Every pair of absolute URI-valued property names **SHALL** be distinct (that is, they **SHALL** differ after normalization) as described in §3.3.2. Similarly, every pair of relative URI-valued property names which lack a uri base id prefix **SHALL** be distinct.

NOTE 3: This restriction ensures that there is only one property in the files object that describes any given physical file.

EXAMPLE 5: This example represents invalid SARIF because the names of two properties in the files object are not distinct; that is, they would be the same if both were normalized.

"files": {

"FILE:///C:/source/input.c": {

...

},

"file:///C:/source/input.c": { # INVALID: the property names are not distinct.

...

}

}

#### Property values

Each property value in the files object **SHALL** be a file object (§3.16) which contains information about the file identified by the property name (§3.11.11.2).

In some cases, a file might be nested within another file (for example, a compressed container), referred to as its “parent.” A file that is not nested within another file is referred to as a “top-level file”. A file that is nested within another file is referred to as a “nested file”.

If the file is a nested file, then the property name **SHALL** specify the URI of the outermost parent, together with a fragment that describes the nesting of the file within its parent or parents. The fragment **SHALL** begin with a forward slash character (“/”), to emphasize that it represents the complete path to the nested file within its container.

EXAMPLE 1: Valid: The fragment begins with a forward slash:

"files": {

"file:///C:/bin/archive.zip#/images/grape.jpg": {

...

}

}

EXAMPLE 2: Invalid: The fragment does not begin with a forward slash:

"files": {

"file:///C:/bin/archive.zip#images/grape.jpg": { # INVALID

...

}

}

If the file is nested more than one level deep in the outermost parent, the fragments representing each level of nesting **MAY** be combined in any way desired, as long as no two of the resulting property names are equivalent as defined in §3.3.2.

NOTE: It does not need to be possible to use this URI to navigate directly to the nested file. The information necessary to do that is specified in the fileLocation property (§3.16.2), or in the offset (§3.16.4) and length (§3.16.5) properties, of each file object.

EXAMPLE 3: Suppose a result is detected within a Flash object contained in a word processing document which is in turn contained in a compressed archive. Suppose the path to the word processing document within the compressed archive is /docs/intro.docx. Then one possible value for the property name within the files object would be:

file:///C:/Code/presentation.zip#/docs/intro.docx/Flash1

If the fragment contains any characters which cannot occur in a fragment as specified in [[RFC3986](#RFC3986)], those character **SHALL** be percent-encoded as specified in [[RFC3986](#RFC3986)].

EXAMPLE 4: Suppose a compressed container contains a file named /docs/chapter#1.doc. Then one possible value for the property name within the files property would be:

file:///C:/Code/presentation.zip#/docs/chapter%231.doc

The “#” character has been percent-encoded as %23.

EXAMPLE 5: This example shows a files property that represents a file nested two levels deep in its outermost container. The first level of nesting is specified by a path within a compressed container. The second level of nesting is specified by a byte offset from the start of the container, together with a length. See §3.16.

"files": {

"file:///C:/Code/app.zip": {

"mimeType": "application/zip",

},

"file:///C:/Code/app.zip#/docs/intro.docx": {

"fileLocation": {

"uri": "/docs/intro.docx",

},

"mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",

"parentKey": "file:///C:/Code/app.zip" # See §3.16.3

},

"file:///C:/Code/app.zip#/docs/intro.docx/Flash1": {

"offset": 17522,

"length": 4050,

"mimeType": "application/x-shockwave-flash",

"parentKey": "file:///C:/Code/app.zip#/docs/intro.docx"

}

}

### logicalLocations property

Depending on the circumstances, a run object either **MAY** or **SHOULD** contain a property named logicalLocations whose value is an object, each of whose properties represents the logical location of one or more results detected in the course of the run.

If the tool has source location information available, and therefore can produce result objects with physical location information (such as the source file name, line, and column), the logicalLocations property **MAY** be present.

If the tool does not have source location information available, and therefore can only produce result objects with logical location information (such as a namespace, type, and method name), the logicalLocations propertys **SHOULD** be present.

With one exception described in §3.20.6, each property name in the logicalLocations object **SHALL** be a string representing the logical location where the result was detected, in a format consistent with the programming language in which the programmatic construct specified by that logical location was expressed. We refer to this string as a “fully qualified logical name”. See §3.20.5 for examples.

Each value in the object specified by the logicalLocations property **SHALL** be a logicalLocation object (§3.23).

In some cases, a logical location might be nested within another logical location (for example, a class nested within a namespace), referred to as its “parent.” A logical location that is not nested within another logical location is referred to as a “top-level logical location”. A logical location that is nested within another logical location is referred to as a “nested logical location”.

If a result is detected in a nested logical location, then the logicalLocations object **SHALL** contain properties describing not only that logical location, but also properties describing each of its parents, up to and including the top-level logical location.

EXAMPLE: In this example, a result was detected in the C++ class namespaceA::namespaceB::classC. The logicalLocations object contains not only a property describing the class, but also properties describing its parents.

"logicalLocations": {

"namespaceA::namespaceB::classC": {

"name": "classC",

"kind": "type",

"parentKey": "namespaceA::namespaceB"

},

"namespaceA::namespaceB": {

"name": "namespaceB",

"kind": "namespace"

"parentKey": "namespaceA"

},

"namespaceA": {

"name": "namespaceA",

"kind": "namespace"

}

}

NOTE: The detailed information in logicalLocations is useful, even though much of it is captured in the location.fullyQualifiedLogicalName property (§3.20.5), because it allows results management systems and other programs to organize analysis results, for example, by asking questions such as “How many results were found in the namespace namespaceA::namespaceB?”. Programs can ask these questions without having to know how to parse the fullyQualifiedLogicalName string.

### results property

A run object **SHALL** contain a property named results whose value is an array containing zero or more result objects (§3.18), each of which represents a single result detected in the course of the run.

NOTE: The results array is not defined to contain unique (§3.6.2) elements because some tools report a line number but not a column number for a result’s location. Such a tool might report the same result twice on the same line, in some cases producing multiple identical result objects.

The results array **SHALL** be empty if the tool invocation that produced the run object did not detect any results.

### resources property

A run object **MAY** contain a property named resources whose value is a resources object (§3.29). A resources object represents items that can be localized, such as resource strings and rule metadata.

### richMessageMimeType property

A run object **MAY** contain a property named richMessageMimeType whose value is a string that specifies the MIME type [[RFC2045](#RFC2045)] of all rich text message properties (§3.9.3) in the run. If this property is absent, its value **SHALL** be taken to be "text/markdown;variant=GFM". [[RFC7763](#RFC7763)] defines the "text/markdown" media type, and [[RFC7764](#RFC7764)] registers "GFM" as the value of the variant parameter which specifies GitHub-Flavored Markdown [[GFM](#GFM)].

For a discussion of the security implications of expressing rich messages in GFM, see §3.9.3.2.

### properties property

A run object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the run that is not explicitly specified in the SARIF format.

## tool object

### General

A tool object contains information describing the analysis tool that was run.

If another tool post-processes the log file (for example, by removing certain results, or by adding information that was not known to the analysis tool), the post-processing tool **SHOULD NOT** alter any part of the tool object.

EXAMPLE:

{

"name": "CodeScanner", # see §3.12.2

"fullName": "CodeScanner 1.1, Developer Preview (en-US)", # see §3.12.3

"semanticVersion": "1.1.2-beta.12", # see §3.12.4

"version": "1.1.2b12, # see §3.12.5

"fileVersion": "1.1.1502.2" # see §3.12.6

}

### name property

A tool object **SHALL** contain a property named name whose value is a string containing the name of the tool that produced the log file.

EXAMPLE: "CodeScanner"

### fullName property

A tool object **MAY** contain a property named fullName whose value is a string containing the name of the tool along with its version and any other useful identifying information, such as its locale.

EXAMPLE: "CodeScanner 1.1, Developer Preview (en-US)"

### semanticVersion property

In a log file produced by an analysis tool, a tool object **SHALL** contain a property named semanticVersion whose value is a string containing the tool version in the format specified by [[SEMVER](#SEMVER)].

EXAMPLE 1:

"tool": {

"semanticVersion": "1.1.2-beta.12"

}

NOTE 1: Semantic versions have the property of being sortable in chronological order of release. The presence of the semanticVersion property allows results management systems to (for example) restrict the results they display to versions newer than a specified version, or to restrict the results to a particular major version.

If the tool does not natively present its version string in SemVer format, it **SHALL** synthesize a SemVer string to populate the semanticVersion property.

EXAMPLE 2: Suppose an analysis tool natively presents its version string as "2.0" (no “patch level” is available). The tool might synthesize a SemVer string "2.0.0".

EXAMPLE 3: Suppose an analysis tool natively presents its version string as "1.1.2b12" (the “pre-release” information is not in SemVer format). The tool might synthesize a SemVer string "1.1.2-beta.12".

A converter **SHALL NOT** emit the semanticVersion property.

NOTE 2: The rationale is that an analysis tool knows whether its version string is intended to be interpreted according to SemVer. A converter will in general not know this, even if the tool's version string conforms to the pattern specified by SemVer.

### version property

In a log file produced by an analysis tool, a tool object **MAY** contain a property named version whose value is a string containing the tool version in whatever format the tool natively provides.

A converter **SHALL** emit the version property.

### fileVersion property

If the operating system on which the tool runs provides a value for the file version of the tool's primary executable file, then the tool object **MAY** contain a property named fileVersion whose value is a string representation of that file version. If the operating system does not provide such a value, the fileVersion property **SHALL** be absent.

EXAMPLE: On the Windows platform, this information is available in the FILEVERSION member of the VERSIONINFO structure.

### language property

A tool object **SHOULD** contain a property named language whose value is a string specifying the language of the messages produced by the tool, in the format specified by [[RFC5646](#RFC5646)]. If this property is absent, its value **SHALL** be taken to be "en-US".

EXAMPLE 1: The tool language is region-neutral English:

"tool": {

"language": "en"

}

EXAMPLE 2: The tool language is French as spoken in France:

"tool": {

"language": "fr-FR"

}

The language property specifies:

1. The language of the message strings contained in the text (§3.9.7) and richText (§3.9.8) properties of any message object (§3.9) in the current run object (§3.11).
2. The language of any embedded resources (§3.9.6) contained in the resources property (§3.11.14) of the current run object.

### resourceLocation property

If a SARIF producer provides external resources (§3.9.6) for languages other than the tool’s declared language (§3.12.7), the tool object **SHALL** contain a property named resourceLocation whose value is a fileLocation object (§3.2) which specifies the location of a directory containing the tool’s SARIF resource files.

If a SARIF producer does not provide external resources, the resourceLocation property **SHALL** be absent.

If the fileLocation object’s uri property (§3.3.2) specifies a relative URI, then its uriBaseId property (§3.3.3) **SHOULD** be present, and the run object’s originalUriBaseIds property (§3.11.10) **SHOULD** contain a property corresponding to the value of the uriBaseId property.

EXAMPLE 1: In this example, a subdirectory of the analysis tool’s installation directory contains the SARIF resource files.

{ # A run object (§3.11).

"tool": {

"name": "SecurityScanner",

"version": "2.0.1",

"resourceLocation": { # A fileLocation object (§3.2).

"uri": "resources",

"uriBaseId": "TOOLINSTALLDIR"

}

},

"originalUriBaseIds": { # See §3.11.10.

"TOOLINSTALLDIR": "file:///C:/Program Files/SecurityScanner/2.0.1"

}

}

EXAMPLE 2: In this example, the SARIF resource files are available on the analysis tool’s web site.

{ # A run object (§3.11).

"tool": {

"name": "SecurityScanner",

"version": "2.0.1",

"resourceLocation": { # A fileLocation object (§3.2).

"uri": ".",

"uriBaseId": "RESOURCES"

}

},

"originalUriBaseIds": { # See §3.11.10.

"RESOURCES": "https://www.example.com/tools/security-scanner/resources/2.0.1"

}

}

If a SARIF producer provides web-based external resources, it **SHOULD** structure its resources directory with subdirectories for each program version, as in EXAMPLE 2 above.

### sarifLoggerVersion property

If the tool that produced the log relied on another software component to generate the log, then the tool object **SHOULD** contain a property named sarifLoggerVersion whose value is a string specifying the version of the logging component.

NOTE: This information is useful, for example, when a tool produces invalid output, and the author of the tool wishes to file a bug report with the author of the logging component. In this case, it is helpful to the author of the logging component to know the precise version number of the logging component that produced the invalid output.

### properties property

A tool object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the themselves that is not explicitly specified in the SARIF format.

## invocation object

### General

An invocation object contains information describing the invocation of the analysis tool that was run.

### commandLine property

An invocation object **MAY** contain a property named commandLine whose value is a string containing the completely specified command line used to invoke the tool, starting with the name of the tool's executable or script file, optionally qualified by the relative or absolute path to the file.

NOTE 1: The information in the commandLine property makes it possible to precisely repeat a run of an analysis tool, and to verify that the results reported in the log file were generated by an appropriate invocation of the tool.

If the information in commandLine contains information which should not be disclosed, such as passwords, tokens, database connection strings, or in some circumstances even the fully qualified path to the tool's executable or script file, that information **SHOULD** be redacted or omitted. Redacted information **SHOULD** be replaced with the token [REMOVED].

NOTE 2: Redacting sensitive information from commandLine makes it more difficult to precisely reproduce an analysis run. The value of commandLine would have to be combined with information from another source to allow the run to be repeated.

EXAMPLE 1: Suppose a tool is invoked with the command line  
  
C:\Users\mary\Tools\DbScanner.exe /ConnectionString  
 "Server=Corp;Db=Accounting;User=Admin;Password=S3cr#t"  
 /input \*.sql  
  
Then the value of the commandLine property might contain the redacted command line  
  
[REMOVED]\DbScanner.exe /connectionString=[REMOVED] /input=\*.sql

The commandLine property might describe a command that would be harmful if it were executed. For this reason, a SARIF consumer that receives of a SARIF log file from an untrusted source **SHOULD NOT** execute the command line without first examining it carefully. In particular, an automated SARIF consumer **SHOULD NOT** execute a command line in a SARIF log file from an untrusted source.

EXAMPLE 2: An example of a harmful command line:

{ # An invocation object

"commandLine": "rm -rf /"

}

### arguments property

An invocation object **MAY** contain a property named arguments whose value is an array of strings, containing in order the command line arguments passed to the tool from the operating system.

EXAMPLE: If the tool is implemented as a C# or Java program, arguments would contain the contents of the args array passed to entry point method.

NOTE: Although the commandLine property (§3.13.2) contains the same information, parsing it is error prone even if one understands the quoting and escaping conventions. SARIF consumers might find the pre-parsed arguments property easier to use.

### responseFiles property

An invocation object **MAY** contain a property named responseFiles whose value is an array of fileLocation objects (§3.3), each of which represents a response file specified on the tool's command line.

A SARIF producer **MAY** embed the contents of a response file in the SARIF log file by mentioning the response file in run.files (§3.11.11), and providing a value for the file.contents property (§3.16.8).

EXAMPLE:

{ # An invocation object.

"commandLine": "/quiet @analyzer.rsp @strict.rsp" @options.rsp,

"responseFiles": [ # An array of fileLocation objects (§3.3).

{ # A fileLocation object.

"uri": "analyzer.rsp",

"uriBaseId": "RESPONSEFILEDIR"

},

{

"uri": "strict.rsp",

"uriBaseId": "RESPONSEFILEDIR"

},

{

"uri": "options.rsp",

"uriBaseId": "RESPONSEFILEDIR"

}

}

...

}

### attachments property

An invocation object **MAY** contain a property named attachments whose value is an array of one or more unique (§3.6.2) attachment objects (§3.14). Each attachment object **SHALL** describe a file relevant to the invocation of the tool. Typically, these would be files specified on the tool’s command line, and therefore mentioned in the commandLine property (§3.13.2) or the arguments property (§3.13.3), if present. They might also be files implicitly consumed by the tool, such as a configuration file.

For an example, see EXAMPLE 1 in §3.14.1.

### startTime property

An invocation object **MAY** contain a property named startTime whose value is a string specifying the date and time at which the run started. The string **SHALL** be in the format specified in §3.8.

### endTime property

An invocation object **MAY** contain a property named endTime whose value is a string specifying the date and time at which the run ended. The string **SHALL** be in the format specified in §3.8.

### exitCode property

If the SARIF producer process did not exit due to a signal, an invocation object **SHOULD** contain a property named exitCode whose value is an integer specifying the process exit code.

If the SARIF producer process exited due to a signal, the exitCode property **SHALL** be absent.

For examples, see §3.13.9.

### exitCodeDescription property

If the SARIF producer process did not exit due to a signal, an invocation object **MAY** contain a property named exitCodeDescription whose value is a string describing the reason for the process exit.

EXAMPLE 1:

{ # An invocation object

"exitCode": 0,

"exitCodeDescription": "Normal successful completion"

}

EXAMPLE 2:

{ # An invocation object

"exitCode": 2,

"exitCodeDescription": "File not found"

}

### exitSignalName property

If the SARIF producer process exited due to a signal, an invocation object **SHOULD** contain a property named exitSignalName whose value is a string containing the name of the signal that caused the process to exit.

If the SARIF producer process did not exit due to a signal, the exitSignalName property **SHALL** be absent.

For an example, see §3.13.11.

### exitSignalNumber property

If the SARIF producer process exited due to a signal, an invocation object **MAY** contain a property named exitSignalNumber whose value is an integer specifying the numeric value of the signal that caused the process to exit.

If the SARIF producer process did not exit due to a signal, the exitSignalNumber property **SHALL** be absent.

EXAMPLE:

{ # An invocation object

"exitSignalNumber": 3,

"exitSignalName": "SIGQUIT"

}

### processStartFailureMessage property

If the analysis tool process failed to start, an invocation object **MAY** contain a property named processStartFailureMessage whose value is a string containing the operating system’s message describing the failure.

NOTE: In this case, the SARIF file would not be produced by the analysis tool (since it failed to start), but rather by some other component of the user’s engineering system which is responsible for monitoring the operation of the analysis tool.

If the analysis tool process started successfully (regardless of whether or how it subsequently failed), the processStartFailureMessage property **SHALL** be absent.

EXAMPLE:

{ # An invocation object

"processStartFailureMessage": "WebScan.exe is not recognized as a command."

}

### toolExecutionSuccessful property

An invocation object **SHOULD** contain a property named toolExecutionSuccessful whose value is a Boolean that is true if the engineering system that started the process knows that the analysis tool succeeded, and false if the engineering system knows that the tool failed. This property is needed because not all programs exit with an exit code of 0 on success and non-0 on failure.

If this property is absent, its value **SHALL** be taken to be false if the exitCode property (§3.13.8) is present and has a non-zero value; otherwise its value **SHALL** be taken to be true.

EXAMPLE:

{

"exitCode": 1,

"exitCodeDescription": "Scan successful; warnings detected.",

"toolExecutionSuccessful": true

}

### machine property

An invocation object **MAY** contain a property named machine whose value is a string containing the name of the machine on which the tool was run.

### account property

An invocation object **MAY** contain a property named account whose value is a string containing the name of the account under which the tool was run.

### processId property

An invocation object **MAY** contain a property named processId whose value is an integer containing the id of the process in which the tool was run.

### fileName property

An invocation object **MAY** contain a property named fileName whose value is a string containing the fully qualified path name of the tool's executable file.

NOTE 1: This property is defined in the invocation object rather than in the tool object (§3.12) because the identical tool might be invoked from different paths on different machines.

NOTE 2: This property might duplicate information in the commandLine property (§3.13.2). It is necessary because the command line might not explicitly specify the path to the tool (for example, if the tool directory is on the execution path), and this information is important for troubleshooting.

NOTE 3: Absolute path names can reveal information that might be sensitive.

### workingDirectory property

An invocation object **MAY** contain a property named workingDirectory whose value is a string containing the fully qualified path name of the directory in which the analysis tool was invoked.

NOTE: Absolute path names can reveal information that might be sensitive.

### environmentVariables property

An invocation object **MAY** contain a property named environmentVariables whose value is an object. The property names in this object **SHALL** contain the names of all the environment variables in the tool's execution environment. The value of each property **SHALL** be a string containing the value of the specified environment variable. If the value of the environment variable is an empty string, the value of the corresponding property **SHALL** be an empty string.

NOTE 1: Environment variable names and values are likely to reveal highly sensitive information. For example, on a Windows machine, environment variables reveal the directories on the execution path, user account name, machine name, logon domain controller, *etc.*

NOTE 2: The result of setting an environment variable to an empty string is operating system-dependent. On Windows, it removes the variable from the environment. In Unix, an environment variable can have an empty value.

### toolNotifications property

A configuration object **MAY** contain a property named toolNotifications whose value is an array of zero or more notification objects (§3.35). Each element of the array represents a runtime condition detected by the invoked process. The presence within this array of any notification object whose level property (§3.35.7) is "error" **SHALL** mean that the run failed.

The information in toolNotifications is primarily intended for the developers of the analysis tool, to aid them in diagnosing bugs in the tool. This contrasts with the information in results, which is intended for the developers of the code being analyzed. However, viewers **MAY** still present tool notifications to users, so users are aware of any tool problems. At a minimum, viewers **SHOULD** make users aware of tool notifications whose level property is "error".

NOTE: Depending on the nature of the error, a tool that encounters a runtime error might or might not be able to continue running.  
  
If the error occurs in the course of evaluating a rule, the tool might report the error in toolNotifications, disable the rule, and continue to execute the remaining rules.  
  
If the error occurs outside of the evaluation of a rule, the tool might report the error in toolNotifications and then halt. If the tool exits abnormally, it might not have the opportunity to report the error.

### configurationNotifications property

A configuration object **MAY** contain a property named configurationNotifications whose value is an array of zero or more notification objects (§3.35). Each element of the array represents a condition relevant to the tool's configuration. The presence within this array of any notification object whose level property (§3.35.7) is "error" **SHALL** mean that the run failed.

The information in configurationNotifications is primarily intended for the engineers who configure the analysis tool, to aid them in diagnosing errors in the configuration. This contrasts with the information in results, which is intended for the developers of the code being analyzed. However, viewers **MAY** still present configuration notifications to users, so users are aware of any configuration problems. At a minimum, viewers **SHOULD** make users aware of configuration notifications whose level property is "error".

NOTE: Many tools can be parameterized with information about which rules to run, and how those rules should be configured. In some cases, if the configuration information is invalid, the tool can ignore the invalid information and continue to run.

EXAMPLE 1: A tool is invoked with a configuration file which specifies that the tool should disable rule ABC0001, but there is no rule whose id is ABC0001. The tool should report the problem in configurationNotifications. The tool might continue to run, reporting results for the rules that are correctly configured.

"configurationNotifications": [

{

"id": "UnknownRule",

"ruleId": "ABC0001",

"level": "warning",

"message": {

"text": "Could not disable rule \"ABC0001\"

because there is no rule with that id."

}

]

EXAMPLE 2: A tool is invoked with an unknown command-line argument. The tool should report the problem in configurationNotifications. The tool might report the problem as a warning and continue to run, or it might report the problem as an error and terminate.

"configurationNotifications": [

{

"id": "UnknownCommandLineArgument",

"level": "error",

"message": {

"text": "Command line argument \"/X\" is unknown."

}

}

]

EXAMPLE 3: A tool is invoked with a command-line argument that specifies the name of a directory containing files to analyze, but the user who invoked the tool does not have read access to that directory. The tool should report the problem as an error in configurationNotifications and then terminate.

"configurationNotifications": [

{

"id": "CannotFindRulePlugin",

"level": "error",

"message": {

"text": "Cannot find rule plugin \"C:\\AnalysisTool\\CustomChecks.dll."

}

}

]

### stdin, stdout, stderr, and stdoutStderr properties

An invocation object **MAY** contain any or all of the properties stdin, stdout, stderr, and stdoutStderr, whose values are physicalLocation objects (§3.21) referring to files that contain the input to and output from the SARIF producer process. stdin, stdout, and stderr refer, respectively, to files containing the contents of the standard input, standard output, and standard error streams. stdoutStderr refers to a file containing the interleaved contents of the standard output and standard error streams. This is useful when the output of those two streams was written to the same file by means of command shell redirection syntax such as "> output.txt 2>&1".

As with any physicalLocation object, if the region property (§3.21.4) is omitted, the object refers to the entire file.

A SARIF producer **MAY** embed the stream contents in the log file by mentioning the corresponding file in the run.files dictionary (§3.11.11), and providing a value for the file.contents property (§3.16.8).

### properties property

An invocation object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the tool invocation that is not explicitly specified in the SARIF format.

## attachment object

### General

An attachment object describes a file relevant to the invocation of a tool (see §3.13.5) or to the detection of a result (see §3.18.17).

A SARIF producer **MAY** embed the contents of an attachment in the log file by mentioning the attachment file in the run.files dictionary (§3.11.10) and providing a value for the file.contents property (§3.16.8).

EXAMPLE 1: In this example, .scanrc is the configuration file for the tool being run:

{ # A run object (§3.11).

"invocations": [ # See §3.11.8.

{ # An invocation object (§3.13).

...

"attachments": [ # See §3.13.5.

{ # An attachment object.

"description": { # See §3.14.2.

"text": "Configuration file"

},

"fileLocation": { # See §3.14.3.

"uri": "file:///C:/Users/Mary/.scanrc"

}

}

]

}

]

}

EXAMPLE 2: In this example, image001.png is a screen shot of the program being analyzed at the point where the result was detected. Note that this example is more appropriate to a dynamic analysis tool than to a static analysis tool.

{ # A result object (§3.18).

...

"attachments": [ # See §3.18.17.

{ # An attachment object.

"description": { # See §3.14.2.

"text": "Screen shot"

},

"fileLocation": { # See §3.14.3.

"uri": "file:///C:/ScanOutput/image001.png"

}

}

]

}

### description property

An attachment object **SHOULD** contain a property named description whose value is a message object (§3.9) describing the role played by the attachment.

### fileLocation property

An attachment object **SHALL** contain a property named fileLocation whose value is a fileLocation object (§3.2) that specifies the location of the attachment file.

## conversion object

### General

A conversion object describes how a converter transformed the output of an analysis tool from the analysis tool’s native output format into the SARIF format.

EXAMPLE: In this example, a converter has converted an AndroidStudio output file into a SARIF log file:

{

...

"runs": [

{

"tool": {

"name": "AndroidStudio"

},

"conversion": {

"tool": { # see §3.15.2

"name": "SARIF SDK Multitool",

},

# see §3.15.3

"invocation": "Sarif.Multitool.exe convert -t AndroidStudio northwind.log"

"analysisToolLogFileLocation": { # see §3.15.4

"uri": "northwind.log",

"uriBaseId": "$LOG\_DIR$"

}

},

"results": [

...

]

}

]

}

### tool property

A conversion object **SHALL** contain a property named tool whose value is a tool object (§3.12) that describes the converter.

### invocation property

A conversion object **MAY** contain a property named invocation whose value is an invocation object (§3.13) that describes the invocation of the converter.

### analysisToolLogFileLocation property

A conversion object **MAY** contain a property named analysisToolLogFileLocation whose value is a fileLocation object (§3.2) that specifies the location of the analysis tool log file that the converter transformed into the SARIF format.

If the analysis tool produced multiple output files, and the converter used more than one of them to synthesize the SARIF results, then the converter **SHOULD NOT** supply a value for this property. Instead, it **MAY** supply a value for the analysisToolLogFileLocation property (§3.19.4) in each analysisToolLogFileContents object (§3.19) in the conversionProvenance property (§3.18.17) of each result object (§3.18).

## file object

### General

A file object represents a single file.

### fileLocation property

Depending on the circumstances, a file object either **SHALL**, **MAY**, or **SHALL NOT** contain a property named fileLocation whose value is a fileLocation object (§3.2).

If the file object represents a top-level file, then the fileLocation property **MAY** be present. If it is present, the value of its uri property (§3.3.2) **SHALL** be equal to the name of the property within run.files (§3.11.10) whose value is this file object. If it is absent, it **SHALL** be interpreted as being present and having a uri property with that same value.

If the file object represents a nested file whose location relative to its parent can be expressed only by means of a path, then the fileLocation property **SHALL** be present, and the value of its uri property **SHALL** be a valid relative URI expressing that path.

If the file object represents a nested file whose location within its parent can be expressed only by a byte offset from the start of the parent, and not by means of a path, then the fileLocation property **SHALL** be absent.

If the file object represents a nested file whose location within its parent can be expressed either by means of a path or by means of a byte offset from the start of the parent, then either the fileLocation property or the offset property (§3.16.4) or both **SHALL** be present; they **SHALL NOT** both be absent. If the fileLocation property is present, the value of its uri property **SHALL** be a valid relative URI expressing the path of the nested file within the parent.

EXAMPLE 1: The fileLocation.uri property of the top-level file repeats the property name. The fileLocation.uri property of the nested file specifies the relative URI of the nested file with respect to its parent.

"files": {

"http://www.example.com/a.zip": {

"fileLocation": {

"uri": "http://www.example.com/a.zip"

},

"mimeType": "application/zip"

},

"http://www.example.com/a.zip#/src/file.c": {

"fileLocation": {

"uri": "/src/file.c"

},

"mimeType": "x-c",

"parentKey": "http://www.example.com/a.zip" # See §3.16.3

}

}

EXAMPLE 2: The fileLocation property of the top-level file is omitted. It is interpreted as being present and having a uri property with the value "http://www.example.com/a.zip".

"files": {

"http://www.example.com/a.zip": {

"mimeType": "application/zip"

},

"http://www.example.com/a.zip#/src/file.c": {

"fileLocation": {

"uri": "/src/file.c"

},

"mimeType": "x-c",

"parentKey": "http://www.example.com/a.zip"

}

}

The value of the fileLocation.uri property for a nested file does not need to match the value of the fragment portion of the URI specified in the property name. This allows multiple levels of nesting to be represented.

EXAMPLE 3: There are two levels of nesting. The fileLocation.uri property of the most deeply nested file does not match the fragment portion of the URI specified in the property name.

"files": {

"http://www.example.com/a.zip": {

"mimeType": "application/zip"

},

"http://www.example.com/a.zip#/media/b.zip": {

"fileLocation": {

"uri": "/media/b.zip"

},

"mimeType": "application/zip",

"parentKey": "http://www.example.com/a.zip"

},

"http://www.example.com/a.zip#/media/b.zip/images/c.png": {

"fileLocation": {

"uri": "/images/c.png"

},

"mimeType": "image/png",

"parentKey": "http://www.example.com/a.zip#/media/b.zip"

}

}

### parentKey property

If the file represented by the file object is a nested file, then the file object **SHALL** contain a property named parentKey whose value is a string containing a URI that matches the property name of the parent file's file object within run.files (§3.11.10).

If the file represented by the file object is a top-level file, then the parentKey property **SHALL** be absent.

NOTE: The presence of the parentKey property makes it possible to navigate from the file object representing a nested file to the file objects representing each of its parent files in turn, up to the top-level file. It is necessary because the URI specified by a file object's property name within run.files does not necessarily contain enough information to do so.

### offset property

Depending on the circumstances, a file object either **SHALL**, **MAY**, or **SHALL NOT** contain a property named offset whose value is a non-negative integer.

If the file object represents a top-level file, then the offset property **SHALL** be absent.

If the file object represents a nested file whose location relative to its parent can be expressed only by means of a byte offset from the start of its parent file, then the offset property **SHALL** be present, and its value **SHALL** be that byte offset.

If the file object represents a nested file whose location within its parent can only be expressed by means of a path, and not by means of a byte offset from the start of the parent, then the offset property **SHALL** be absent.

If the file object represents a nested file whose location within its parent can be expressed either by means of a path or by means of a byte offset from the start of the parent, then either the fileLocation property (§3.16.2) or the offset property or both **SHALL** be present; they **SHALL NOT** both be absent. If the offset property is present, its value **SHALL** be that byte offset.

### length property

A file object **MAY** contain a property named length whose value is a non-negative integer specifying the length of the file in bytes.

### mimeType property

A file object **SHOULD** contain a property named mimeType whose value is a string that specifies the MIME type [[RFC2045](#RFC2045)] of the file.

### hashes property

A file object **MAY** contain a property named hashes whose value is an array of unique (§3.6.2) hash objects (§3.17), each of which specifies a hashed value for the file specified by the file object, along with the name of the algorithm used to compute the hash.

If present, the array specified by hashes **SHALL NOT** be empty.

NOTE: A hash value for an analysis target can be useful when a log file is processed by a result management system. The value can be used as a key when persisting results in a database. This allows a build system to use cached results, rather than repeating the analysis, when a target has not changed. A file hash can also be useful for validating results in a policy compliance system, allowing an auditor to validate that rerunning analysis against a target that hashes to a specific value reproduces the provided results.  
  
The file object defines an array of hash values, rather than a single hash value, to allow a log file to be consumed by multiple tool chains that might expect hash values produced by differing algorithms. Compliance systems, for example, will favor the use of secure hash algorithms (such as SHA-256) that minimize the possibility that two different targets will produce the same hash (at the expense of speed to produce the hash). In situations where compliance and security are not a concern, a system might prefer to use a fast hash algorithm (such as MD5 or SHA-1) even though they have known weaknesses that allow adversaries to more easily generate hash collisions.  
  
To populate the hashes property, an analysis tool needs the ability to produce hashes for its analysis targets. Alternatively, the hashes could be added to the log file as a post-processing step.  
  
To make the best use of such an analysis tool, a user (such as a build engineer) would determine what systems in their build environment will consume the log file. The user would then configure the tool to produce hashes using the algorithms required by those systems. Analysis tools that are configurable to produce hashes with a variety of commonly used algorithms will interoperate most easily with such systems.

### contents property

A file object **MAY** contain a property named contents whose value is a fileContent object (§3.2) representing the entire contents of the file.

### properties property

A file object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the file that is not explicitly specified in the SARIF format.

## hash object

### General

A hash object represents a hash value of some file or collection of files, together with the algorithm used to compute the hash.

EXAMPLE:

{

"value":"b13ce2678a8807ba0765ab94a0ecd394f869bc81", # see §3.17.2

"algorithm":"sha256" # see §3.17.3

}

### value property

A hash object **SHALL** contain a property named value whose value is a string representation of the hash value of some file or collection of files, computed by the algorithm named in the algorithm property (§3.17.3).

NOTE: The value is represented as a string because hash values are typically represented in hexadecimal notation, and JSON integer values must be decimal.

### algorithm property

A hash object **SHALL** contain a property named algorithm whose value is a string specifying the name of the algorithm used to compute the hash value specified in the value property (§3.17.2). This string **SHALL** be one of the following:

* "authentihash"
* "blake256"
* "blake512"
* "ecoh"
* "fsb"
* "gost"
* "groestl"
* "has160"
* "haval"
* "jh"
* "md2"
* "md4"
* "md5"
* "md6"
* "radioGatun"
* "ripeMD"
* "ripeMD128"
* "ripeMD160"
* "ripeMD320"
* "sdhash"
* "sha1"
* "sha224"
* "sha256"
* "sha384"
* "sha512"
* "sha3"
* "skein"
* "snefru"
* "spectralHash"
* "ssdeep"
* "swifft"
* "tiger"
* "tlsh"
* "whirlpool"

## result object

### General

A result object describes a single result detected by an analysis tool.

### Constraints

At least one of the message property (§3.18.7) and the ruleMessageId property (§3.18.8) **SHALL** be present. If they are both present, and they both refer to message strings that are present in the log file, then those message strings **SHALL** be identical.

EXAMPLE 1: In this example, result.message.text (§3.9.7) directly contains the message string "Variable n is uninitialized.". The result.ruleId property (§3.18.4) and the result.ruleMessageId property (§3.18.8) together designate the rule message string "Variable {0} is uninitialized." which, along with the contents of the result.message.arguments property (§3.9.11), produces the identical string.

{ # A run object (§3.11).

"results": [ # See §3.11.13.

{ # A result object (§3.18).

"ruleId": "CA2101", # See §3.18.4.

"message": { # See §3.18.7.

"text": "Variable n is uninitialized.", # See §3.9.7.

"arguments": [ "n" ] # See §3.9.11.

},

"ruleMessageId": "default" # See §3.18.8.

}

],

"resources": { # See §3.11.14.

"rules": { # See §3.29.3

"CA2010": { # A rule object (§3.30).

"messageStrings": { # See §3.30.7.

"default": "Variable {0} is uninitialized."

}

}

}

}

}

EXAMPLE 2: In this example, the SARIF log file does not include rule metadata. The SARIF log file is valid even though the external resource string (§3.9.6) designated by ruleId and ruleMessageId might not produce the same string as message.text.

{ # A run object (§3.11).

"results": [ # See §3.11.13.

{ # A result object (§3.18).

"ruleId": "CA2101", # See §3.18.4.

"message": { # See §3.18.7.

"text": "Variable n is uninitialized.", # See §3.9.7.

"arguments": [ "n" ] # See §3.9.11.

},

"ruleMessageId": "default" # See §3.18.8.

}

]

}

### id property

A result object **MAY** contain a property named id whose value is a string that uniquely identifies the result over time.

This property can be used in a variety of ways, for example:

* A SARIF producer could synthesize a unique identifier such as a GUID and set this property to that value. A result management system could use that value as a database key. In this scenario, the result management system might store multiple instances of what are logically the same result.
* A SARIF producer could compute a full fingerprint for the issue and set this property to that value. Again, a result management system could use that value as a database key. In this scenario, the result management system could take note when an already-existing issue is stored in it.
* A SARIF producer could refrain from setting this property, optionally supply a value for the fingerprintContributions property (§3.18.11), and allow the result management system to compute a full fingerprint. Then, if a consumer retrieved results in SARIF format from the result management system, the result management system could populate this property with the full fingerprint.

For more information on the computation and usage of result fingerprints, see Appendix B, “Use of fingerprints by result management systems.”

### ruleId property

Depending on the circumstances, a result object either **SHALL** or **SHALL NOT** contain a property named ruleId whose value is a string containing the stable, opaque identifier for the rule that was evaluated to produce the result.

EXAMPLE:

"results": [

{

"ruleId": "CA2101"

...

}

]

Direct producers **SHALL** emit the ruleId property.

Not all existing analysis tools emit the equivalent of a ruleId in their output. A converter which converts the output of such an analysis tool to the SARIF format **SHALL NOT** set the ruleId property, and in particular, it **SHALL NOT** attempt to synthesize it from other information available in the original analysis tool's output.

### ruleKey property

If there is more than one rule with the id specified by the ruleId property (§3.18.3), and if the run object (§3.11) in which this result occurs contains a resources.rules property (§3.11.14, §3.29.3), then the result object **SHALL** contain a property named ruleKey whose value is a string that matches one of the property names in the resources.rules object.

The value of the ruleId property on this result object **SHALL** match the id property (§3.30.3) of the rule object (§3.29) identified by ruleKey.

EXAMPLE: In this example, there is more than one rule with id CA1711. When the log includes a result with that rule id, it provides a value for ruleKey to specify which of the rules with that id is meant.

"runs": [

{

"results": [

{

"ruleId": "CA1711", # Matches the "id" value of the specified

# property value within "rules"

"ruleKey": "CA711-1" # Specifies a property name within "rules".

}

],

"resources": {

"rules": {

"CA1711-1": {

"id": "CA1711"

},

"CA1711-2": {

"id": "CA1711"

}

}

}

}

]

### level property

A result object **MAY** contain a property named level whose value is one of a fixed set of strings that specify the severity level of the result.

If present, the level property **SHALL** have one of the following values, with the specified meanings:

* "pass": The rule specified by the ruleId property (§3.18.3) was evaluated, and no problem was found.
* "warning": The rule specified by the ruleId property was evaluated, and a problem was found.
* "error": The rule specified by the ruleId property was evaluated, and a serious problem was found.
* "open": The rule specified by the ruleId property was evaluated, and the tool concluded that there was insufficient information to decide whether a problem exists.
* "notApplicable": The rule specified by the ruleId property was not evaluated, because it does not apply to the file specified by analysisTarget (§3.20.3).

EXAMPLE 1: In this example, a binary checker has a rule that applies to 32-bit binaries only. It produces a "notApplicable" result if it is run on a 64-bit binary:

"results": [

{

"ruleId": "ABC0001",

"level": "notApplicable",

"message": {

"text": "\"MyTool64.exe\" was not evaluated for rule ABC0001

because it is not a 32-bit binary."

},

"locations": [

{

"analysisTarget": {

"uri": "file://C:/bin/MyTool64.exe"

}

}

]

}

]

* "note": A purely informational log entry.  
    
  The ruleId property for a result object whose level property is "note" **MAY** be present, if the note relates to a particular rule; otherwise ruleId **MAY** be absent.

EXAMPLE 2: In this example, the tool reports an observation about the code that does not represent a problem.

"results": [

{

"ruleId": "ABC0002",

"level": "note",

"message": {

"text": "Consider using 'nameof(start)' instead of hard-coding

the parameter name 'start'."

},

"locations": [

{

"analysisTarget": {

"uri": "file:///C:/code/a.cs",

"region": {

"startLine": 6

}

}

}

]

}

]

EXAMPLE 3: In this example, the tool reports information that is relevant to a particular rule but does not represent an observation about the code.

"results": [

{

"ruleId": "ABC0003",

"level": "note",

"message": {

"text": "A new version of rule ABC0003 is available."

}

}

]

EXAMPLE 4: In this example, the tool reports information that is not related to any rule and is not an observation about the code.

"results": [

{

"level": "note",

"message": {

"text": "Version 11.0 of SuperLint is now available."

}

}

]

If the level property is absent, its value **SHALL** be taken to be the value of the defaultLevel property (§3.31.3) of the ruleConfiguration object (§3.31) contained in the configuration property (§3.30.11) of the rule object (§3.30) specified by this result object's ruleId property (§3.18.3) or ruleKey property (§3.18.5).

In that case, if the run object (§3.11) containing this result does not include a resources.rules property (§3.11.14, §3.29.3) (and no external resource file is available), or if the resources.rules property does not specify information for the rule object associated with this result, or if the rule object associated with this result does not specify a configuration.defaultLevel property, then the value of the level property **SHALL** be taken to be "warning".

### message property

A result object **MAY** contain a property named message whose value is a message object (§3.9) that describes the result. If the message property is absent, the ruleMessageId property (§3.18.8) **SHALL** be present. Both message and ruleMessageId **MAY** be present. See §3.18.2 for more information.

The message property **SHOULD** provide sufficient details to allow an end user to resolve any problem that the result might indicate. In particular, it **SHALL** include all of the following information that is available and relevant to the result:

* Information sufficient to identify the analysis target, and the location within the target where the problem occurred.
* The condition within the analysis target that led to the problem being reported.
* The risks potentially associated with not fixing the problem.
* The full range of responses to the problem that the end user could take (including the definition of conditions where it might be appropriate not to fix the problem, or to conclude that the result is a false positive).

EXAMPLE: This is an example of a message:

"results": [

{

"message": {

"text": "Deleting member 'x' of variable 'y' may compromise

performance on subsequent accesses of 'y'. Consider

setting object member 'x' to null instead, unless this

object is a dictionary or if runtime semantics otherwise

dictate that the existence of a null member is distinct

from one that is not present at all. This violation can

also be ignored for infrequently called code paths."

}

}

]

### ruleMessageId property

A result object **MAY** contain a property named ruleMessageId whose value is a string that identifies the message within the rule metadata for the rule used in this result. If the ruleMessageId property is absent, the message property (§3.18.7) **SHALL** be present. Both message and ruleMessageId **MAY** be present. See §3.18.2 for more information.

The value of ruleMessageId **SHALL** correspond to one of the property names in the messageStrings property (§3.30.7) of the rule object (§3.30) whose id property (§3.30.3) matches the ruleKey property (§3.18.5) (if present) or else the ruleId property (§3.18.4) of the current result object.

The value of ruleMessageId **MAY** also correspond to one of the property names in the richMessageStrings property (§3.30.8) of that rule object.

EXAMPLE: In this example, the result object’s id and ruleMessageId properties together specify the string identified by "default" within the rule metadata for the rule whose id is "CA2101".

{ # A run object (§3.11)

"results": [

{ # A result object (§3.18)

"id": "CA2101",

"ruleMessageId": "default",

...

}

],

"resources": { # A resources object (§3.29)

"rules": {

"CA2101": {

"id": "CA2101",

"messageStrings": {

"default": "This is the default message for this rule.",

"special": "This is another message for this rule, used in special cases"

}

}

}

}

}

### locations property

A result object **SHOULD** contain a property named locations whose value is an array of one or more unique (§3.6.2) location objects (§3.19), each of which specifies a location where the result occurred.

NOTE: In rare circumstances, it might not be possible to specify a location for a result. However, locations is very valuable information for anyone who needs to diagnose and correct the condition described by the result, so the authors of analysis tools should make every effort to provide it.

EXAMPLE 1: If a C++ analyzer detects that no file defines a global function main, then that result cannot be associated with a file.

The locations array **SHALL NOT** contain more than one element unless the condition indicated by the result, if any, can only be corrected by making a change at every location specified in the array.

EXAMPLE 2: In C#, which support “partial” classes, portions of the declaration of a single class can occur at multiple locations in the source code. If an analysis tool reports that the name of such a class does not conform to a specified convention, then the resulting log file should contain a single result object, which should contain a locations array each of whose elements specifies a location in the source code where the class name occurs.

The locations array **SHALL NOT** be used to specify distinct occurrences of the same result, which can be corrected independently.

EXAMPLE 3: Consider an analysis tool which locates misspelled words in documentation, and suppose this tool scans a document in which the same word is misspelled in two distinct locations. Then the resulting log file must contain two distinct result objects, each of which contains a locations array containing a single location object specifying the location of one instance of the misspelled word.  
  
In contrast, consider a tool which locates misspelled words in variable names. If the tool detects a misspelled variable name, it must produce a single result object whose locations array contains the location of every reference to the variable, since fixing some but not all of the references would cause a compilation error.

### snippet property

A result object **MAY** contain a property named snippet whose value is a fileContent object (§3.2) containing a source code or other file fragment that illustrates the result, for example, the text of the source code line on which the result was detected, or a small range of lines surrounding the result location.

### fingerprintContributions property

A result object **MAY** contain a property named fingerprintContributions whose value is a JSON object (§3.5). Each property value in this object **SHALL** be a string that contributes to the unique identity of the result. [Appendix B](#AppendixFingerprints) explains how a result management system can use these values. The property names that identify these values are arbitrary strings. A SARIF producer **MAY** use the property names to identify the nature of the information used to compute the fingerprint contribution.

EXAMPLE 1: In this example, the SARIF producer has computed two fingerprints, one based on a hash of a code snippet that encompasses the region where the result was detected, and one based on hash of the entire contents of the file in which the result was detected.

{ # A result object

...

"fingerprintContributions": {

"snippetHash": "56eaf900cc8f6",

"fileHash": "7eed42b4987dc"

}

}

EXAMPLE 2. In this example, the SARIF producer has computed a single fingerprint. It has chosen an arbitrary value for the corresponding property name.

{ # A result object

...

"fingerprintContributions": {

"1": "56eaf900cc8f6"

}

}

A SARIF consumer **MAY** use any algorithm or heuristic to determine whether two distinct result objects logically represent the same result. For example, it might require a majority of the fingerprints contributions to match. We refer to this algorithm or heuristic as a “result matching procedure.”

To make use of the information, if any, embodied in the property names, a SARIF consumer requires knowledge of the naming convention used by the SARIF producer. A SARIF consumer with that knowledge **MAY** use the property names in its result matching procedure. For example, it might only require fingerprint contributions with certain property names to match. A SARIF consumer lacking that knowledge **SHALL NOT** use the property names in its result matching procedure.

Because SARIF consumers might come to depend on the choice of property names, SARIF producers that use property names to identify the nature of the information used to compute the fingerprint contribution **SHOULD** adhere to the following guidelines:

* Choose meaningful property names that describe the information used to compute the fingerprint contribution.
* Document the property names.
* When introducing a fingerprint contribution computed with a different approach, associate it with a new property name.
* Avoid removing existing property names and fingerprint contributions, since existing SARIF consumers might rely on them.

### codeFlows property

A result object **MAY** contain a property named codeFlows whose value is an array of one or more unique (§3.6.2) codeFlow objects (§3.24). The codeFlows property is intended for use by analysis tools that provide execution path details that illustrate a possible problem in the code. We refer to this execution path as a code flow. Each codeFlow object in the codeFlows array **SHALL** describe a single code flow.

NOTE: The SARIF file format allows multiple codeFlow objectss within a single result object to allow for the possibility that more than one path through the program might be relevant to a single result.

### stacks property

A result object **MAY** contain a property named stacks whose value is an array of one or more unique (§3.6.2) stack objects (§3.25). The stacks property is intended for use by analysis tools that collects call stack information in the process of producing results.

NOTE: The SARIF file format allows multiple stack objects within a single result object to allow for the possibility that more than one call stack might be relevant to a single result.

### relatedLocations property

A result object **MAY** contain a property named relatedLocations whose value is an array of one or more unique (§3.6.2) annotatedCodeLocation objects (§3.27), each of which represents a location relevant to understanding the result.

EXAMPLE: Suppose that a tool for analyzing JavaScript has a rule that reports a problem when a variable declared in an inner scope hides a variable with the same name in an enclosing scope. The tool would report the problem on the line where the inner variable is declared. The tool could choose to add an element to the relatedLocations array, specifying the location where the outer variable was declared.  
  
The result might appear in the log file like this:

results: [

{

"ruleId": "JS3056",

"level": "error",

"message": {

"text": "Name 'index' cannot be used in this scope because

it would give a different meaning to 'index'."

},

"locations": [

{

"analysisTarget": {

"uri": "file:///C:/Code/a.js",

"region": {

"startLine": "6",

"startColumn": "10"

}

}

}

],

"relatedLocations": [ # An array of annotatedCodeLocation objects

# (§3.27)

{

"message": {

"text": "The previous declaration of 'index' was here."

},

"physicalLocation": {

"uri": "file:///C:/Code/a.js",

"region": {

"startLine": "2",

"startColumn": "6"

}

}

}

]

},

...

]

The tool might write messages to the console like this:

C:\Code\a.js(6,10-10) : error : JS3056: Name 'index' cannot be used in this scope because it would give a different meaning to 'index'.

C:\Code\a.js(2,6-6) : info : JS3056: The previous declaration of 'index' was here.

### suppressionStates property

#### General

A result object **MAY** contain a property named suppressionStates whose value is an array of unique (§3.6.2) strings. This property **SHALL** be present if and only if the analysis tool that produced the log file wishes to convey the information that the condition described by the result object should be “suppressed”.

NOTE: The treatment of “suppressed” results depends on the development environment within which the log file is used, for example, a build system, an integrated development environment (IDE), or a result management system. Typically, development environments do not expose suppressed results to the user. For example, they do not include them in build log files, display them in error lists, or include them in bug counts.

If present, this property conveys the reason or reasons that the result has been suppressed. The supported reasons for suppressing a result are:

* The developer has suppressed the result in the source code (see §3.18.15.2).
* The result is marked as suppressed in an external store such as a database (see §3.18.15.3).

#### suppressedInSource value

Some programming languages offer a syntactic construct for suppressing compiler warnings.

EXAMPLE: In C#, #pragma warning is such a construct.

For tools that examine source code written in such a language, the suppressionStates array **SHALL** include the value "suppressedInSource" if the tool determines that the result occurred at a location within the scope of an instance of such a construct which is intended to suppress that particular class of result. If the tool determines that the result did not occur at such a location, or if the tool cannot or chooses not to determine whether the result occurred at such a location, or if the tool examines source code written in a language that lacks such a construct, the suppressionStates array **SHALL NOT** include the value "suppressedInSource".

#### suppressedExternally value

Some development environments provide a persistent store, for example a database, containing historical information about the results from analysis tools. Such a store might offer the ability to mark a result as “suppressed,” meaning that if the result is encountered again, it should be ignored.

When a tool with access to such a database detects such a result, it **MAY** choose not to add the result to the log. If the tool does include such a result in the log, the suppressionStates array **SHALL** include the value "suppressedExternally".

If the tool does not have access to a database of suppression information, or if the tool does have access to such a database and determines that the result is not marked for suppression in that database, then the suppressionStates array **SHALL NOT** include the value "suppressedExternally".

### baselineState property

A result object **MAY** contain a property named baselineState whose value is a string that specifies the state of this result with respect to some previous run. We refer to this previous run as the “baseline”.

If the run.baselineId property (§3.11.4) of the current run is present, the baseline **SHALL BE** the run specified by run.baselineId.

If the run.baselineId property of the current run is absent, then a SARIF consumer will need out of band information available to determine the baseline.

This property **SHALL** have one of the following values, with the specified meanings:

* "new": This result was detected in the current run but was not detected in the baseline.
* "existing": This result was detected both in the current run and in the baseline.
* "absent": This result was detected in the baseline but was not detected in the current run.

If the run.baselineId property is present but the baselineState property is absent, the baselineState property **SHALL** be considered to have the value "new".

NOTE: The purpose of the baselineState property is to allow (for example) a measurement of how many new results were introduced in the run, and how many previously existing results no longer appear.  
  
To assign a value to baselineState, a tool needs a way to determine whether a result is “the same”, in some sense, as a result that appeared in the baseline. [Appendix B](#AppendixFingerprints) discusses how a result management system can assign a “fingerprint” to each result. An analysis tool that works together with such a result management system can use the fingerprint to determine whether two results are the same; two results with the same fingerprint are considered the same.

### attachments property

A result object **MAY** contain a property named attachments whose value is an array of one or more unique (§3.6.2) attachment objects (§3.14). Each attachment object **SHALL** describe a file relevant to the detection of the result.

For an example, see EXAMPLE 2 in §3.14.1.

### conversionProvenance property

If the run object (§3.11) containing a result object was produced by a converter, the result object **MAY** contain a property named conversionProvenance whose value is an array of one or more unique (§3.6.2) analysisToolLogFileContents objects (§3.19) which specify the portions of the analysis tool’s output that the converter transformed into the result object.

Direct producers **SHALL NOT** emit the conversionProvenance property.

The value of this property is an array, rather than a single analysisToolLogFileContents object, because, depending on the analysis tool, the converter might need to bring together information from multiple output files to synthesize the result object.

This property is intended to be useful to developers of converters, to help them debug the conversion from the analysis tool’s native output format into the SARIF format.

For an example, see §3.19.1.

### fixes property

A result object **MAY** contain a property names fixes whose value is an array of one or more unique (§3.6.2) fix objects (§3.31).

### properties property

A result object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the result that is not explicitly specified in the SARIF format.

## analysisToolLogFileContents object

### General

An analysisToolLogFileContents object describes a portion of an analysis tool’s output that a converter transformed into a result object (§3.18). This object is only relevant in a SARIF log file that was produced by a converter. It is intended to be useful to developers of converters, to help them debug the conversion from the analysis tool’s native output format into the SARIF format.

EXAMPLE: Given this Android Studio output file:

<?xml version="1.0" encoding="UTF-8"?>

<problems>

<problem>

<file></file>

<line>242</line>

...

<problem\_class ...>Assertions</problem\_class>

...

<description>Assertions are unreliable. ...</description>

</problem>

</problems>

a SARIF converter might transform it into the following SARIF log file:

{

...

"runs": [

{

"tool": {

"name": "AndroidStudio",

...

},

"conversion": { # A conversion object (see §3.14)

...

},

"results": [

{

"ruleId": "Assertions",

"message": {

"text": "Assertions are unreliable. ..."

},

...

"conversionProvenance": [ # An array of analysisToolLogFileContents objects

{

"region": { # See §3.19.2.

"startLine": 3,

"startColumn": 3,

"endLine": 12,

"endColumn": 13

},

"snippet": { # See §3.19.3.

"text": "<problem>\n ... \n </problem>",

},

"analysisToolLogFileLocation": { # See §3.19.4.

"uri": "AndroidStudio.log",

"uriBaseId": "$LOGSROOT"

}

}

],

...

}

]

}

]

}

### region property

An analysisToolLogFileContents object **MAY** contain a property named region whose value is a region object (§3.22) which specifies the region of the analysis tool’s log file that was transformed into the result object.

If this property is absent, the region **SHALL** be taken to be the entirety of the analysis tool’s log file.

### snippet property

An analysisToolLogFileContents object **MAY** contain a property named snippet whose value is a fileContent object (§3.2) which contains the text of that region of the analysis tool’s log file that was transformed into the result object.

### analysisToolLogFileLocation property

An analysisToolLogFileContents object **MAY** contain a property named analysisToolLogFileLocation whose value is a fileLocation object (§3.2) that specifies the location of one of the analysis tool log files that the converter transformed into the result object that ultimately contains this analysisToolLogFileContents object.

If this property is absent, its value **SHALL** be taken to be the value of the run.conversion.analysisToolLogFileLocation property (§3.15.4), if that property is present.

## location object

### General

A location object specifies the location where an analysis tool detected a result. Depending on the circumstances, a location object specifies the physical location (§3.21) of the result, the logical location (§3.20.5) of the result, or both.

A logical location specifies a programmatic construct, for example, a class name or a function name, without specifying the programming artifact within which that construct occurs.

NOTE: There are two reasons to include logical locations in the SARIF format in addition to physical locations:

1. In the absence of symbol information, binary analysis tools might not have source code locations available, so information about line and column numbers might not be present in the log file. In this case, code editors, other programs, or end users can use logical location to navigate from a result to the correct source code location.  
  
2. Logical location information is an important contributor to fingerprinting scenarios, because it is typically more resilient to changes in source code than are line locations. See [Appendix B](#AppendixFingerprints) for more information about fingerprinting. The fullyQualifiedLogicalName property (§3.20.5) is particularly convenient for fingerprinting.

### Constraints

Depending on the information available to the tool that SARIF producer, either or both of the analysisTarget property (§3.20.3) and the resultFile property (§3.20.4) **SHALL** be present.

If the SARIF producer file knows the analysis target, then the analysisTarget property **SHALL** be present. If the SARIF producer knows that the result file is different from the analysis target, then the resultFile property **SHALL** be present; otherwise the resultFile property **SHALL** be absent.

NOTE: Generally, an analysis tool will know both the file it was instructed to scan (the analysis target) and the file in which it detects a problem (the result file).

EXAMPLE 1: Suppose an analysis tool for C++ source code is instructed to scan the source file *a.cpp*, and suppose the tool detects a problem in *a.cpp*. In this case, the tool should set the analysisTarget property to *a.cpp*, and it should not set the resultFile property.

EXAMPLE 2: Suppose an analysis tool for C++ source code is instructed to scan the source file *a.cpp*, which includes the header file *b.h*, and suppose the tool detects a problem in *b.h*. In this case, the tool should set the analysisTarget property to *a.cpp*, and it should set the resultFile property to *b.h*.

EXAMPLE 3: Suppose an analysis tool for object code detects a problem in the binary file *c.dll*, and suppose the tool has available symbol information which maps that location within the binary to a specific line in a source file *d.cpp*. In this case, the tool should set the analysisTarget property to *c.dll*, and it should set the *resultFile* property to *d.cpp*.

If the SARIF producer does not know the analysis target, then the resultFile property **SHALL** be present and the analysisTarget property **SHALL** be absent.

NOTE: Some analysis tools produce output in a format that does not include both the analysis target and the result file. In such cases, a converter which translates the output into the SARIF format might only have the result file available.

EXAMPLE 4: Suppose an analysis tool for C++ source code is instructed to scan the source file *a.cpp*, which includes the header file *b.h*, and suppose the tool detects a problem in *b.h*. Suppose further that the tool produces output in a format other than SARIF, for example:  
  
{ "file": "b.h", "line": 6, "col": 1, "msg": "Uninitialized" }  
  
Suppose a converter attempts to translate this output into SARIF format. Suppose that the converter does not know whether the analysis tool was instructed to scan a source file that included *b.h*, or whether it was instructed to scan *b.h* directly. In this case, the converter only knows that the problem occurred in *b.h*. The converter should set the resultFile property to *b.h*, and it should not set the analysisTarget property.

### analysisTarget property

Depending on the information available to the SARIF producer (see §3.20.2), a location object either **SHALL** or **SHALL NOT** contain a property named analysisTarget whose value is a physicalLocation object (§3.21) which identifies the file that the analysis tool was instructed to scan. This does not need to be the same as the file where the result actually occurred.

### resultFile property

Depending on the information available to the SARIF producer (see §3.20.2), a location object either **SHALL** or **SHALL NOT** contain a property named resultFile whose value is a physicalLocation object (§3.21) that identifies the file where the analysis tool detected the result.

### fullyQualifiedLogicalName property

Depending on the circumstances, a location object either **SHOULD** or **MAY** contain a property named fullyQualifiedLogicalName whose value is a string which specifies the fully qualified name of the logical location where the analysis tool detected the result. If physical location information is not available, fullyQualifiedLogicalName **SHOULD** be present. Otherwise, fullyQualifiedLogicalName **MAY** be present.

The format of the fullyQualifiedLogicalName string **SHALL** be consistent with the programming language in which the programmatic construct specified by that logical location was expressed.

EXAMPLE 1: C: create\_process

EXAMPLE 2: C++: Namespace1::Class::Method(int, double) const &&

EXAMPLE 3: C#: Namespace1.Class.Method(string, int[])

If the run.logicalLocations property (§3.11.12) is present, the value of the fullyQualifiedLogicalName property **SHALL** be equal to the name of one of the properties on the run.logicalLocations object, with one exception, described in §3.20.6.

NOTE: There are a few reasons the fullyQualifiedLogicalName property exists, even though the information it contains is presented in more detail in the run.logicalLocations property:

* It allows a log file viewer to display the logical location in a way that is easily understood by users.
* As mentioned in §3.20.1, fullyQualifiedLogicalName is also particularly convenient for fingerprinting, although the more detailed information in run.logicalLocations could be used instead.
* It relieves viewers from having to format the logical location from the more detailed information in run.logicalLocations.
* It is useful for producing readable in-source suppressions (for example, “suppress all instance of rule CA2101 in the class NamespaceA.NamespaceB.ClassC”).

### logicalLocationKey property

A location object **MAY** contain a property named logicalLocationKey whose value is a string. If present, this string **SHALL** be equal to the name of one of the properties on the run.logicalLocations object (§3.11.12), which provides additional information about the logical location specified by fullyQualifiedLogicalName (§3.20.5).

logicalLocationKey is only necessary if, in the course of a run, the tool produces results in two or more distinct logical locations with the same fullyQualifiedLogicalName. In that case, the tool **SHALL** synthesize a unique name by appending a suffix to fullyQualifiedLogicalName, assign the resulting string to logicalLocationKey, and use that string as the key into the run.logicalLocations dictionary.

EXAMPLE: Suppose a tool analyzes two C++ source files:

// file1.cpp

namespace A {

class B {

}

}

// file2.cpp

namespace A {

namespace B {

class C {

}

}

}

(These could not coexist in the same compilation, but there is no reason two such source files could not exist.)

If the tool detected one result in class B in *file1.cpp*, and another result in namespace B in *file2.cpp*, the fullyQualifiedLogicalName for both would be A::B. In that case, the tool might set the logicalLocationKey property in either one of the results to A::B-1, and it might populate the logicalLocations property as follows:

"logicalLocations": {

"A::B": [

{

"name": "A",

"kind": "namespace"

},

{

"name": "B",

"kind": "namespace"

}

],

"A::B-1": [

{

"name": "A",

"kind": "namespace"

},

{

"name": "B",

"kind": "type"

}

]

}

### decoratedName property

A location object **MAY** contain a property named decoratedName whose value is a string containing the compiler's internal representation of the logical location associated with this location object.

Even though decoratedName describes a logical location, the presence of decoratedName does not require that fullyQualifiedLogicalName (§3.20.5) also be present.

EXAMPLE: In this example, the decoratedName property contains a “mangled” name emitted by a C++ compiler:

{ # A "location" object

"fullyQualifiedLogicalName": "b::c(float)",

"decoratedName": "?c@b@@AAGXM@Z"

}

### properties property

A location object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the location that is not explicitly specified in the SARIF format.

## physicalLocation object

### General

A physicalLocation object represents the physical location where a result was detected. A physical location specifies a reference to a programming artifact together with a region within that artifact.

### id property

A physicalLocation object **MAY** contain a property named id whose value is a non-negative integer that **SHALL** be unique among all physicalLocation objects belonging to the result object (§3.18) within which it occurs. The value does not need to be unique across all result objects in the run.

EXAMPLE: Within a result object, the following property values (among others) are physicalLocation objects, and no two of them can have the same values for their id properties:

result.relatedLocations[0].physicalLocation

result.codeFlows[0].locations[0].physicalLocation

result.stacks[0].frames[0].physicalLocation

The purpose of the id property is to enable an embedded link (§3.9.4) within a message to refer to the location. If no message within the current result object refers to this location *via* an embedded link, the id property does not need to appear.

### fileLocation property

With certain exceptions, a physicalLocation object **SHALL** contain a property named fileLocation whose value is fileLocation object (§3.2) that represents the location of the file.

The exceptions are as follows:

* Under certain circumstances, if the physicalLocation object appears as the value of an annotatedCodeLocation.targetLocation property (§3.27.10), the fileLocation property **MAY** be absent, as described in §3.27.10.
* Under certain circumstances, if the physicalLocation object appears as a member of an annotation.locations array (§3.28.3) which in turn appears as the value of an annotatedCodeLocation.annotations property (§3.27.15), the fileLocation property **MAY** be absent, as described in §3.27.15.

If the run.files property (§3.11.10) is present, the value of the fileLocation.uri property **SHOULD** be equal to the name of one of the properties on the run.files object, which provides additional information about the file specified by fileLocation.

EXAMPLE: In this example, the value of the property runs[0].results[0].locations[0].analysisTarget.fileLocation.uri is equal to the name of the property runs[0].files[0][file:///C:/Code/main.c].

{

"version": "2.0",

"runs": [

{

"files": {

"file:///C:/Code/main.c": [

{

"mimeType": "text/x-c",

}

]

},

"results": [

{

"ruleId": "CA2101",

"level": "error",

"locations": [

{

"analysisTarget": {

"fileLocation": {

"uri": "file:///C:/Code/main.c"

},

"region: {

"startLine": 24,

"startColumn": 9

}

}

}

]

}

]

}

]

}

### region property

A physicalLocation object **MAY** contain a property named region whose value is a region object (§3.22) that represents the region within a file where the result was detected.

If the result occurs in a nested file, then the region property **SHALL** specify the location of the result with respect to the innermost nested file.

EXAMPLE: If a result occurs in a C++ file contained in a compressed archive, then the region would represent the line and column number of the result with the C++ file. It would not represent (for example) the offset of the C++ file from the start of the archive.

If the region property is absent, the physicalLocation object refers to the entire file.

## region object

### General

A region object represents a region, that is, a contiguous portion of a file. Every property in a region object is represented by a non-negative integer, that is, by a JSON number value with no sign, no fractional part, and no exponent part.

SARIF defines two types of regions: text regions and binary regions.

* A text region represents a contiguous range of zero or more characters.
* A binary region represents a contiguous range of zero or more bytes.

SARIF defines different properties to represent text regions and binary regions.

In a text region, the startLine property (§3.22.4) **SHALL** be present and **SHALL** have a value greater than 0. In a binary region, the startLine property **SHALL** be absent.

NOTE 1: SARIF consumers can use the presence or absence of the startLine property to determine whether to treat a region as a text region or as a binary region.

NOTE 2: It is up to each analysis tool whether to treat a given file as a text file (in which case it should emit text regions for results detected in the file) or as a binary file (in which case it should emit binary regions).

### Text regions

The line number of the first line in a text file **SHALL** have the value 1. The column number of the first character in each line **SHALL** have the value 1.

NOTE 1: SARIF defines column number as a count of characters. If a line in a text file contains tab characters, viewers may choose to present column numbers that match the visual offset of each character from the beginning of the line. These “visual” column numbers will not match the column numbers contained in the SARIF file.

Depending on the file's character encoding, each character might be represented by one byte or by multiple bytes. In source files encoded in UTF-16, characters outside the Basic Multilingual Plane (BMP) are represented as a sequence of two 16-bit code points; this sequence is called a “surrogate pair” [[UNICODE10](#UNICODE10)]. Tools that report results in UTF-16-encoded files **SHALL** consider characters outside the BMP as occupying two columns.

NOTE 2: The reason for this requirement is that is common for existing tools to ignore surrogate pairs when calculating column numbers.

Programs such as viewers that process SARIF log files together with the analysis target files to which those log files refer **SHOULD** attempt to determine the character encoding of the target files. In the absence of internal information such as a Byte Order Mark, viewers **MAY** use external information (for example, command line arguments, project settings, or other configuration information) to determine the character encoding. If external information is also lacking, viewers **SHOULD** assume that each character occupies one byte.

The start of a text region **SHALL** be represented by a combination of the startLine (§3.22.4) and startColumn (§3.22.5) properties. startLine **SHALL** be present. If startColumn is absent, the region **SHALL** be considered to start at column 1. For the remainder of this section, whenever startColumn is mentioned, it includes the case where startColumn is absent and so is taken to be 1.

The end of a text region **SHALL** be represented either by a combination of the endLine (§3.22.6) and endColumn (§3.22.7) properties, or by the length property (§3.22.9).

If endLine is absent and endColumn is present, endLine **SHALL** be taken to be the same as startLine.

If endLine is present and endColumn is absent, then:

* If endLine is the same as startLine, then endColumn **SHALL** be taken to be the same as startColumn.
* If endLine is different from startLine, then endColumn **SHALL** be taken to be 1.

For the remainder of this section, whenever endLine is mentioned, it includes the case where endLine is absent and so is taken to be the same as startLine.

For the remainder of this section, whenever endColumn is mentioned, it includes the case where endColumn is absent and so has its default value, which depends on the value of endLine as described above.

If endLine is the same as startLine and endColumn is the same as startColumn, the length of the region **SHALL** be taken to be 0.

If length is present, it **SHALL** be non-negative and **SHALL** represent a count of characters.

If none of endLine, endColumn, or length is present, the length of the region **SHALL** be taken to be 0.

endLine **SHALL** be greater than or equal to startLine.

If endLine is equal to startLine, then endColumn **SHALL** be greater than or equal to startColumn.

To represent a region that includes the last character in a line, excluding any trailing newline sequence, endColumn **SHALL** be set to a value 1 greater than the number of characters in the line, excluding the newline sequence if present. This is the case even for the last line of the file, which might not end with a newline sequence.

EXAMPLE 1: Suppose a text file contains the following line, on line 5:

abcde

Then the region with startLine = 5, startColumn = 3, endLine = 5, and endColumn = 6 represent the three characters cde. This is the case whether or not the line ends with a newline sequence.

To include a newline sequence in a region, endLine **SHALL** be greater than startLine.

EXAMPLE 2: Suppose a text file contains the following lines, starting on line 5:

abcde

fg

Then the region with startLine = 5, startColumn = 3, endLine = 6, and endColumn = 1 represent the three characters cde plus a newline sequence.

### Binary regions

The start of a binary region **SHALL** be represented by the offset property (§3.22.8), which denotes the offset in bytes from the start of the file. The offset of the first byte in a file **SHALL** have the value 0.

The end of a binary region **SHALL** be represented by the length property (§3.22.9), which denotes a count of bytes. If length is absent, the length of the region **SHALL** be taken to be 0.

In a binary region, the startLine (§3.22.4), startColumn (§3.22.5), endLine (§3.22.6), and endColumn (§3.22.7) properties **SHALL** be absent.

### startLine property

When a region object represents a text region, it **SHALL** contain a property named startLine, which **SHALL** have an integer value equal to the line number of the line containing the first character in the region.

The line number of the first line in the file **SHALL** be 1.

### startColumn property

When a region object represents a text region, it **MAY** contain a property named startColumn, which **SHALL** have an integer value equal to the column number of the first character in the region.

The column number of the first column on each line **SHALL** be 1.

If startColumn is absent, it **SHALL** be inferred as specified in §3.22.2.

### endLine property

When a region object represents a text region, it **MAY** contain a property named endLine, which **SHALL** have an integer value equal to the line number of the line containing the last character in the region.

If endLine is absent, it **SHALL** be inferred as specified in §3.22.2.

### endColumn property

When a region object represents a text region, it **MAY** contain a property named endColumn, which **SHALL** have an integer value equal to the column number of the last character in the region.

If endColumn is absent, it **SHALL** be inferred as specified in §3.22.2.

### offset property

When a region object represents a binary region, it **SHALL** contain a property named offset, which **SHALL** have a non-negative integer value equal to the byte offset from the beginning of the file of the first byte in the region.

When a region object represents a text region, the offset property **MAY** be present. In this case, it **SHALL** represent the character offset from the beginning of the file of the first character in the region.

### length property

A region object **MAY** contain a property named length whose value is a non-negative integer.

When the region object represents a text region, the value of length **SHALL** be the number of characters in the region. If the region consists of 0 characters, then either length **SHALL** either be absent or if **SHALL** have the value 0.

When a region object represents a binary region, the value of length **SHALL** be the number of bytes in the region. If the region consists of 0 bytes, then either length **SHALL** either be absent or if **SHALL** have the value 0.

The sum of the offset (§3.22.8) and length properties **SHALL** be greater than or equal to 0, and less than or equal to the length the file, which is measured in characters for a text region and in bytes for a binary region.

A region whose offset is equal to the length of the file and whose length is 0 permitted, and **SHALL** represent an insertion point at the end of the file.

## logicalLocation object

### General

A logicalLocation object describes a logical location.

logicalLocation objects occur as property values within the run.logicalLocations object (§3.11.12).

### name property

A logicalLocation object **SHALL** contain a property named name whose value is a string that identifies the construct in which the result occurred. For example, this property might contain the name of a class or a method.

The name property does need to be suitable for display.

EXAMPLE 1: A C++ analysis tool might emit the name property of a function as the “decorated” function name, which encodes the function signature in a manner that is compiler-dependent and not easily readable.

If the logicalLocation object describes a top-level logical location, and if the name property would be equal to the name of the property for which this object provides the value, then the name property **MAY** be absent.

EXAMPLE 2: In this example, the logical location is a top-level C++ function named functionF, and name is omitted.

"logicalLocations": {

"functionF": {

"kind": "function"

}

}

EXAMPLE 3: In this example, the logical location is a top-level C++ function, and name is equal to the property name.

"logicalLocations": {

"functionF": {

"name": "functionF",

"kind": "function"

}

}

EXAMPLE 4: In this example, the logical location is a top-level C++ function, but name is not equal to the property name, so it cannot be omitted.

"logicalLocations": {

"functionF-0": {

"name": "functionF",

"kind": "function"

}

}

### kind property

A logicalLocation object **SHOULD** contain a property named kind whose value is one of the following strings, if any of those strings accurately describes the construct identified by this object:

* "function"
* "member"
* "module"
* "namespace"
* "package"
* "resource"
* "type"

If none of those strings accurately describes the construct, kind **MAY** contain any value specified by the analysis tool.

### parentKey property

If the logical location represented by the logicalLocation object is a nested logical location, then the logicalLocation object **SHALL** contain a property named parentKey whose value is a string that matches the property name of the parent logicalLocation object within run.logicalLocations (§3.11.12).

If the logical location represented by the logicalLocation object is a top-level logical location, then the parentKey property **SHALL** be absent.

## codeFlow object

### General

A code flow is a sequence of locations that specify a possible execution path through the code.

### message property

A codeFlow object **MAY** contain a property named message whose value is a message object (§3.9) relevant to the code flow.

### locations property

A codeFlow object **SHALL** contain a property named locations whose value is an array of one or more annotatedCodeLocation objects (§3.27). Each element of the array **SHALL** represent a single location visited by the tool in the course of producing the result. This array does not need to include every location visited by the tool, but the elements that are present **SHALL** occur in the order that the tool visited them. The elements do need to be unique within the array.

NOTE: The locations array might include multiple identical elements if, for example, the analysis tool simulated the execution of a loop in the course of producing the result.

### properties property

A codeFlow object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the code flow that is not explicitly specified in the SARIF format.

## stack object

### General

A stack object describes a single call stack. A call stack is a sequence of nested function calls, each of which is referred to as a stack frame.

### message property

A stack object **MAY** contain a property named message whose value is message object (§3.9) relevant to this call stack.

### frames property

A stack object **SHALL** contain a property named frames whose value is an array of one or more stackFrame objects (§3.26). This array **SHALL** include every function call in the stack for which the tool has information, and the entries that are present **SHALL** occur in chronological order with the most recent (innermost) call first and the least recent (outermost) call last. The entries in this array do not need to be unique within the array.

NOTE 1: It is possible for the same frame to occur multiple times if the call stack includes a recursion.

NOTE 2: It is possible that the analysis tool will not have location information for every frame in the call stack. This might happen if, for example, application code for which location information is available calls into operating system code for which location information is not available, which in turn calls back into application code.

### properties property

A stack object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the stack that is not explicitly specified in the SARIF format.

## stackFrame object

### General

A stackFrame object describes a single stack frame within a call stack (§3.25).

### message property

A stackFrame object **MAY** contain a property named message whose value is a message object (§3.9) relevant to this stack frame.

### physicalLocation property

A stackFrame object **MAY** contain a property named physicalLocation whose value is a physicalLocation object (§3.21) specifying the location to which this stack frame refers.

### module property

A stackFrame object **MAY** contain a property named module whose value is a string containing the name of the module that contains the location to which this stack frame refers.

### threadId property

A stackFrame object **MAY** contain a property named threadId whose value is an integer which identifies the thread on which the code at the location specified by this object was executed.

### fullyQualifiedLogicalName property

A stackFrame object **SHALL** contain a property named fullyQualifiedLogicalName whose value is a string containing the fully qualified name of the method to which this stack frame refers. See §3.20.5 for examples.

If the run.logicalLocations property (§3.11.12) is present, the value of the fullyQualifiedLogicalName property **SHOULD** be equal to the name of one of the properties on the run.logicalLocations object, with one exception, described in §3.26.7.

### logicalLocationKey property

A stackFrame object **MAY** contain a property named logicalLocationKey whose value is a string. If present, this string **SHALL** be equal to the name of one of the properties on the run.logicalLocations object (§3.11.12), which provides additional information about the logical location specified by fullyQualifiedLogicalName (§3.26.6).

logicalLocationKey is only necessary if, in the course of a run, the tool produces results in two or more distinct logical locations with the same fullyQualifiedLogicalName. In that case, the tool **SHALL** synthesize a unique name by appending a suffix to fullyQualifiedLogicalName, assign the resulting string to logicalLocationKey, and use that string as the key into the run.logicalLocations dictionary.

### address property

A stackFrame object **MAY** contain a property named address whose value is a non-negative integer containing the address in memory of the location represented by this stack frame.

### offset property

A stackFrame object **MAY** contain a property named offset whose value is a non-negative integer containing the byte offset of the location represented by this stack frame from the start of the method represented by this stack frame.

NOTE: This is distinct from the physicalLocation.region.offset property (§3.22.8), if any, specified by the physicalLocation property (§3.26.3). physicalLocation.region.offset specifies an offset from the start of a file, not from the start of a method.

### parameters property

A stackFrame object **MAY** contain a property named parameters whose value is an array of strings representing the parameters of the function call represented by this stack frame.

### properties property

A stackFrame object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the stack frame that is not explicitly specified in the SARIF format.

## annotatedCodeLocation object

### General

An annotatedCodeLocation object represents a physical location together with additional information relevant to the use of the location in a particular context.

### step property

If an annotatedCodeLocation object occurs within a codeFlow, it **MAY** contain a property named step. If the annotatedCodeLocation does not occur within a codeFlow, the step property **SHALL** be absent.

The value of the step property **SHALL** be an integer whose value is the 1-based sequence number of the location within the code flow, that is, it **SHALL** be 1 for the first location, 2 for the second, and so on.

NOTE: This property has two primary purposes:

* A viewer can display the identifier next to each location when it displays a code flow.
* A user reading the log file can easily refer to the location in conversation, for example, “I think the problem occurs at step 6.”

### physicalLocation property

An annotatedCodeLocation object **SHOULD** contain a property named physicalLocation whose value is a physicalLocation object (§3.21) that specifies the file location to which the annotatedCodeLocation object refers.

This property **SHOULD NOT** be absent unless the tool does not have physical location information for this annotatedCodeLocation.

NOTE: This could happen if, for example:

* This annotatedCodeLocation refers to a location within a binary for which the tool does not have associated symbol information.
* This annotatedCodeLocation occurs within a codeFlow (§3.24), the value of the kind property (§3.27.9) is "functionExit", and the tool has chosen not to associate the function exit with a source code location.
* This annotatedCodeLocation occurs within a codeFlow, the value of the kind property is "continuation", and the continuation is used purely to record a change to global state (which might happen asynchronously with respect to the code flow).

### fullyQualifiedLogicalName property

Depending on the circumstance, an annotatedCodeLocation object either **SHOULD** or **MAY** contain a property named fullyQualifiedLogicalName whose value is a string containing the fully qualified name of the method to which this annotatedCodeLocation refers. If the physicalLocation property (§3.27.3) is absent, fullyQualifiedLogicalName **SHOULD** be present. Otherwise, fullyQualifiedLogicalName **MAY** be present. See §3.20.5 for examples.

If the run.logicalLocations property (§3.11.12) is present, the value of the fullyQualifiedLogicalName property **SHOULD** be equal to the name of one of the properties on the run.logicalLocations object, with one exception, described in §3.27.5.

### logicalLocationKey property

An annotatedCodeLocation object **MAY** contain a property named logicalLocationKey whose value is a string. If present, this string **SHALL** be equal to the name of one of the properties on the run.logicalLocations object (§3.11.12), which provides additional information about the logical location specified by fullyQualifiedLogicalName (§3.27.4).

logicalLocationKey is only necessary if, in the course of a run, the tool produces results in two or more distinct logical locations with the same fullyQualifiedLogicalName. In that case, the tool **SHALL** synthesize a unique name by appending a suffix to fullyQualifiedLogicalName, assign the resulting string to logicalLocationKey, and use that string as the key into the run.logicalLocations dictionary.

### module property

An annotatedCodeLocation object **MAY** contain a property named module whose value is a string containing the name of the module that contains the code location specified by this object.

### threadId property

An annotatedCodeLocation object **MAY** contain a property named threadId whose value is an integer which identifies the thread that was executing when the execution of a code flow reached the location specified by this object. If this annotatedCodeLocation does not occur within a codeFlow, the threadId property **SHALL** be absent.

### message property

An annotatedCodeLocation object **MAY** contain a property named message whose value is a message object (§3.9) that describes the significance of this location within a particular context.

### kind property

An annotatedCodeLocation object **MAY** contain a property named kind whose value is a string that categorizes the location.

If present, the kind property **SHALL** have one of the following values, with the specified meanings:

* "alias": This location defines an additional name for a variable defined in a declaration.
* "assignment": At this location, an assignment to a variable occurred.
* "branch": At this location, a branch in the execution path occurred.
* "call": This location is the site of a function or method call. Every annotatedCodeLocation whose kind property is "call" **SHALL** be paired with a subsequent annotatedCodeLocation whose kind property is "callReturn" and which refers to the same function, unless the codeFlow in which the call occurs terminates before the function returns.
* "callReturn": This location is the target of a return from a function or method.

NOTE 1: Viewers can use the "call" and "callReturn" values to clarify the presentation of a code flow that crosses function boundaries. For example, when displaying the list of locations in a code flow, a viewer could indent the locations between a "call" and a "callReturn".

* "continuation": Execution continued at this location.

NOTE 2: This can be used, for example, to designate the target of a jump instruction, or the statement after the end of a loop.

* "declaration": The location introduces into the program a name which denotes an entity such as a variable, function, template, *etc.*
* "functionEnter": This location is an entry point to a function or method. Every annotatedCodeLocation whose kind property is "functionEnter" **SHALL** be paired with a subsequent annotatedCodeLocation whose kind property is "functionExit" and which refers to the same function, unless the codeFlow in which the call occurs terminates before a function exit point is reached.
* "functionExit": This location represents the conceptual exit from the function, used by some analysis tools to represent the final node in the directed acyclic graph that represents the control flow through a function. A "functionExit" **MAY** be preceded in the code flow by a "functionReturn".

NOTE 3: A tool might choose (for example) to associate a "functionExit" with the closing brace of a function, or to associate it with the final statement in the function, or not to associate it with a source code location at all.

* "functionReturn": This is the location of a statement that returns control from a function or method (for example, a return statement).
* "usage": At this location, data is used.

NOTE 4: Some analysis tools track the usage of *untrusted* data.

EXAMPLE: Suppose an analysis tool produces a result which states that a piece of data from an insecure source has been used at a particular location. The tool might provide a “related location” (§3.18.14) whose value is an annotatedCodeLocation object with the message “Insecure data entered the system here”.

### kind-dependent properties: target, targetLocation, values and state

Depending on the value of its kind property (§3.27.9), an annotatedCodeLocation object either **MAY**, **SHOULD**, or **SHALL NOT** contain:

* a property named target whose value is a string.
* a property named targetLocation whose value is a physicalLocation object (§3.21).
* a property named values whose value is an array of strings.
* a property named state whose value is an object.

These properties **SHALL** appear only in annotatedCodeLocation objects that are part of a codeFlow (§3.24).

The precise interpretation of these properties, and whether they **MAY**, **SHOULD**, or **SHALL NOT** be present, depends on the value of the kind property.

NOTE 1: In imprecise terms, the meanings of these properties are as follows:

* target represents the thing being operated on at the specified location.
* targetLocation represents the physical location of that thing.
* values represents a set of values that are input to the operation or produced by the operation.
* state is a set of key/value pairs, each of which represents a variable or expression which participates in the operation.

If both the targetLocation property and the physicalLocation property (§3.27.3) of this annotatedCodeLocation object are present, then targetLocation.uri (§3.21.3) **MAY** be absent, in which case it is considered to have the same value as physicalLocation.uri.

The format of the string value of the target property, the elements of the values array, the property names in the state object, and the property values in the state object, **SHALL** be consistent with the syntax of the programming language in which the code being analyzed was written.

In this section, a “variable name” **MAY** be any of the following, unless otherwise specified:

* A simple variable name.
* An object property reference.
* An array element reference.
* A reference to the current object.
* Any combination of these.
* Any valid expression that produces a value.

EXAMPLE 1: Examples of valid “variable names” in C++:

* count
* str->length
* values[0]
* this
* this->size
* this->car->wheels[0]
* func() (assuming that func returns a value)

In this section, whenever a “value” is mentioned, it means a string representation of the value.

EXAMPLE 2: Examples of valid “values”:

* A integer value of 2 would be represented as "2".
* A string value of "2" would be represented as "\"2\"".
* A Boolean value of true would be represented as "true".

NOTE 2: In languages where all objects have a built-in string representation (for example, by means of a method such as ToString()), the analysis tool might choose to obtain the string representation by calling that method. For example, in C#, given an object uri of type System.Uri, the tool might choose to obtain the string value by calling uri.ToString(), perhaps resulting in "http://www.example.com".

The requirements and interpretation of the target, targetLocation, values, and state properties are as follows:

* When kind is "alias":
  + target **SHOULD** be present. If present, its value **SHALL** be the name of the alias being created. If multiple aliases are created in the same source language statement, the analysis tool **SHALL** create a separate annotatedCodeLocation object for each alias that the tool wishes to represent in the log.
  + targetLocation **SHALL** be absent.
  + values **SHOULD** be present. If present, its value **SHALL** be an array with one element, whose value is the name of the variable being aliased.
  + state **MAY** be present. If present, it **SHALL** contain a single property whose name is the name of the variable being aliased, and whose value is the value of that variable.
* When kind is "assignment":
  + target **SHOULD** be present. If present, its value **SHALL** be the name of the variable being assigned to. If multiple variables are assigned to in the same source statement, the analysis tool **SHALL** create a separate annotatedCodeLocation object for each assignment that the tool wishes to represent in the log.
  + targetLocation **SHALL** be absent.
  + values **SHOULD** be present. If present, its value **SHALL** be an array with one element, whose value is the value assigned to the target variable.
  + state **MAY** be present. If present, it **SHALL** contain properties which specify the names and values of selected variables or subexpressions which participate in the expression on the right-hand side of the assignment.
* When kind is "branch":
  + target **SHOULD** be present if the target of the branch is a named label, in which case its value **SHALL** be the name of the label; otherwise, it **SHALL** be absent.
  + targetLocation **MAY** be present. If present, its value **SHALL** specify the location of the target of the branch.
  + values **MAY** be present if the branch is the result of a test, in which case its value **SHALL** be an array with one element, whose value is the Boolean value of the test condition; otherwise, it **SHALL** be absent.
  + state **MAY** be present if the branch is the result of a test, in which case it **SHALL** contain properties which specify the names and values of selected variables or subexpressions which participate in the expression being tested; otherwise, it **SHALL** be absent.
* When kind is "call":
  + target **SHOULD** be present. If present, its value **SHALL** be the fully qualified name of the function being called.
  + targetLocation **MAY** be present. If present, its value **SHALL** specify the physical location of the function being called.
  + values **MAY** be present. If present, its value **SHALL** be an array containing the values of the arguments level
  + object reference (for example, this) passed to object method calls.
  + state **MAY** be present. If present, it **SHALL** contain properties which specify the names and values of selected variables or subexpressions participating in the expressions passed as arguments to the function. For object method calls, this **MAY** include the name and value of the object on which the method was invoked, or any variables or subexpressions which participate in an expression which resolves to that object.
* When kind is "callReturn":
  + target **SHOULD** be present. If present, its value **SHALL** be the fully qualified name of the function being returned from.
  + targetLocation **SHALL** be absent.
  + values **MAY** be present, in which case its value **SHALL** be an array containing the value or values returned from the function; otherwise, it **SHALL** be absent.
  + state **MAY** be present. If present, it **SHALL** contain the names and values of any parameters that were passed by reference to the called function and whose value was reassigned by the called function.
* When kind is "continuation":
  + target **SHALL** be absent.
  + targetLocation **SHALL** be absent.
  + values **SHALL** be absent.
  + state **MAY** be present. If present, it **SHALL** contain the names and values of selected variables or expressions at the specified location. Any variable that is in scope at the specified location **MAY** be mentioned or used in an expression.
* When kind is "declaration":
  + target **SHOULD** be present. If present, its value **SHALL** be the name of the variable being declared. If multiple variables are declared in the same source statement, the analysis tool **SHALL** create a separate annotatedCodeLocation object for each declaration that the tool wishes to represent in the log.
  + targetLocation **SHALL** be absent.
  + values **MAY** be present if the declaration has an initializer, in which case its value **SHALL** be an array containing one element, whose value **SHALL** be the value of the initializer expression, or if the variable is automatically initialized to a default value, in which case its value **SHALL** be an array containing one element, whose value **SHALL** be that default value; otherwise, it **SHALL** be absent.
  + state **MAY** be present if the declaration has an initializer, in which case it **SHALL** contain the names and values of selected variables or subexpressions participating in the initializer expression; otherwise, it **SHALL** be absent.
* When kind is "functionEnter":
  + target **SHOULD** be present. If present, its value **SHALL** be the fully qualified name of the function being entered. If there is a matching "functionExit", then either both of them or neither of them **SHALL** specify target, and if they do, their values **SHALL** be the same.
  + targetLocation **SHALL** be absent.
  + values **MAY** be present. If present, its value **SHALL** be an array containing the values of the arguments passed to the function. This array **SHALL NOT** include the implicit object reference (for example, this) passed to object method calls.
  + state **MAY** be present. If present, it **SHALL** contain the names and values of selected variables or expressions at the specified location. Any variable whose value is available at the specified location **MAY** be mentioned or used in an expression.
* When kind is "functionExit":
  + target **SHOULD** be present. If present, its value **SHALL** be the fully qualified name of the function being returned from. If there is a matching "functionEnter", then either both of them or neither of them **SHALL** specify target, and if they do, their values **SHALL** be the same.
  + targetLocation **SHALL** be absent.
  + values **MAY** be present if the function returns a value or values, in which case its value **SHALL** be an array containing the value or values returned from the function; otherwise, it **SHALL** be absent.
  + state **SHALL** be absent.
* When kind is "functionReturn":
  + target **SHOULD** be present. If present, its value **SHALL** be the fully qualified name of the function being returned from. If there is a matching "functionEnter", then either both of them or neither of them **SHALL** specify target, and if they do, their values **SHALL** be the same.
  + targetLocation **SHALL** be absent.
  + values **MAY** be present if the function returns a value or values, in which case its value **SHALL** be an array containing the value or values returned from the function; otherwise, it **SHALL** be absent.
  + state **MAY** be present if the function returns a value or values, in which case it **SHALL** contain properties which specify the names and values of selected variables or subexpressions which participate in the expressions which produce the returned value or values; otherwise, it **SHALL** be absent.
* When kind is "usage":
  + target **SHOULD** be present. If present, its value **SHALL** be the name of the variable being used. If multiple variables are used in the same source statement, the analysis tool **SHALL** create a separate annotatedCodeLocation object for each usage that the tool wishes to represent in the log.
  + targetLocation **SHALL** be absent.
  + values **MAY** be present. If present, its value **SHALL** be an array with one element, whose value is the value of the used variable at the specified location.
  + state **SHALL** be absent.

EXAMPLE 3: In C++, if the source code contains the declaration

std::string &str = name;

then the value of kind should be "alias", the value of target should be "str", the value of values should be

[ "name" ]

and the value of state might be

{ "name": "\"John\"" }

EXAMPLE 4: In C++, if the source code contains the declaration

std::string &str = name, &str2 = address;

and if the tool creating the log wished to represent both aliases in the log file, then the tool should create two annotatedCodeLocation objects, each with kind set to "alias", and referring to the same source line.

EXAMPLE 5: In C++ or C#, if the source code contains the assignment

m = n + p;

then the value of kind should be "assignment", the value of target should be "m", the value of values might be

[ "5" ]

and the value of state might be

{ "n": "2", "p": "3" }

Or, since state can include expressions, the value of state might be

{ "n + p": "5" }

or even

{ "n": "2", "p": "3", "n + p": "5" }

EXAMPLE 6: In C#, if the source code contains the test

if (s.Length > 0 && y > 2 && valid())

then the value of kind should be "branch", target should be absent, the value of values might be

[ "true" ]

and the value of state might be

{ "s": "\"A string\"", "y": "3" }

or perhaps

{ "s": "\"A string\"", "s.Length": "8", "y": "3", "valid()": "true" }

EXAMPLE 7: In C++ or C#, if the source code contains the function call

func(7, m + n, "s", this, g(2));

then the value of kind should be "call", the value of target might be "func" (or, for example, "N.C.func" if the function func occurred in class C in namespace N), the value of values should be

[ "7", "m + n", "\"s\"", "this", "g(2)" ]

and the value of state might be

{ "m": "2", "n": "3" }

If present, the value of targetLocation would be the physical location where func is defined.

EXAMPLE 8: In C#, if the source code contains the method invocation

example.Func(n);

where example is an object of type SomeClass, then the value of kind should be "call", the value of target should be "SomeClass.Func", the value of values might be

[ "5" ]

and the value of state might be

{ "example": "null", "n": "5" }

(assuming that the method was mistakenly invoked on a null reference).

EXAMPLE 9: In C++ or C#, if the source code contains the function call:

int n = func();

then the value of kind should be "callReturn", the value of target might be "func" (or, for example, "N.C.func" if the function func occurred in class C in namespace N), the value of values might be

[ "5" ]

(assuming that the function returned the value 5), and state should be absent.

EXAMPLE 10: In C++ or C#, if the source code contains the declaration

int m = n + p;

then the value of kind should be "declaration", the value of target should be "m", the value of values might be

[ "5" ]

and the value of state might be

{ "n": "2", "p": "3" }

EXAMPLE 11: In C++ or C#, if the source code contains the declaration

int m = n + p, q = k + r;

and if the tool creating the log wished to represent the declarations of both variables in the log file, then the tool should create two annotatedCodeLocation objects, each with kind set to "declaration", and referring to the same source line.

EXAMPLE 12: In C++ or C#, if the source code contains the return statement

int func()

{

...

return m + n;

}

then the value of kind should be "functionExit", the value of target might be "func" (or, for example, "N.C.func" if the function func occurred in class C in namespace N), the value of values might be

[ "5" ]

and the value of state might be

{ "m": "2", "n": "3" }

If the run.logicalLocations property (§3.11.12) is present, and the value of kind is "call", then the value of the target property **SHOULD** be equal to the name of one of the properties on the run.logicalLocations object, with one exception, described in §3.27.11.

### targetKey property

The annotatedCodeLocation object **MAY** contain a property named targetKey whose value is a string. If present, this string **SHALL** be equal to the name of one of the properties on the run.logicalLocations object (§3.11.12), which provides additional information about the function specified by target (§3.27.10).

targetKey is only necessary if, in the course of a run, the tool encounters two or more distinct functions with the same fully qualified logical name. In that case, the tool **SHALL** synthesize a unique name by appending a suffix to target, assign the resulting string to targetKey, and use that string as the key into the run.logicalLocations dictionary.

### importance property

An annotatedCodeLocation object **MAY** contain a property named importance whose value is a string that specifies the importance of this annotatedCodeLocation in understanding the codeFlow object (§3.24) in which it occurs. If this annotatedCodeLocation does not occur within a codeFlow, the importance property **SHALL** be absent.

If present, the importance property **SHALL** have one of the following values, with the specified meanings:

* "important": this location is important for understanding the code flow.
* "essential": this location is essential for understanding the code flow.
* "unimportant": this location contributes to a more detailed understanding of the code flow, but is not normally needed.

If this property is absent, it **SHALL** be considered to have the value "important".

NOTE: A viewer might use this property to offer the user three options for viewing a lengthy code flow:

* A “normal view,” which omits locations whose importance property is "unimportant".
* An “abbreviated view,” which displays only those locations whose importance property is "essential".
* A “verbose view,” which displays all the locations in the code flow.

### taintKind property

An annotatedCodeLocation object **MAY** contain a property named taintKind whose value is a string which classifies state transitions in code locations relevant to a taint analysis.

If present, the taintKind property **SHALL** have one of the following values, with the specified meanings:

* "source": At this location, untrusted data enters the system (for example, by being provided by a user or read from a file on disk).
* "sanitizer": This is the location of a statement (for example, a function call), after the execution of which data that entered the system from outside (for example, from user input) is presumed to be safe.
* "sink": At this location, untrusted data enters some security-sensitive code (for example, an eval statement that converts untrusted text to executable code).

### snippet property

An annotatedCodeLocation object **MAY** contain a property named snippet whose value is a fileContent object (§3.2) containing the text of the source code lines specified by annotatedCodeLocation.physicalLocation.region (§3.21.4).

### annotations property

An annotatedCodeLocation object **MAY** contain a property named annotations whose value is an array containing one or more unique (§3.6.2) annotation objects (§3.28), each of which describes one or more additional physical locations which are relevant to this annotatedCodeLocation object.

EXAMPLE: Consider an annotatedCodeLocation object which describes the declaration statement

int x = (y + z) \* q;

The kind property should be "declaration", the target property should be "x", the values property might be "42", and the state property might be

{ "y": "2", "z": "4", "y + z": "6", "q": "7" }

Now, if the analysis tool wanted to emphasize the value of the expression (y + z), for example, to allow a viewer to highlight the expression, or to display a message when the mouse hovered over the expression, it might set the annotations property to

[ # An array of annotation objects

{ # An annotation object (§3.28)

"message": {

"text": "(y + z) = 42"

},

"locations": [ # An array of physicalLocation objects

{ # A physicalLocation object (§3.21)

# The fileLocation property can be omitted if it is

# the same as annotatedCodeLocation.physicalLocation

# .fileLocation.

"region": {

"startLine": 12,

"startColumn": 13,

"endColumn": 19

}

}

]

}

]

For any integer array indices i and j, if value the of the property annotatedCodeLocation.annotations[i].locations[j].fileLocation is the same as the value of the property annotatedCodeLocation.physicalLocation.fileLocation, then the fileLocation property **MAY** be omitted from the physicalLocation object annotatedCodeLocation.annotations[i].locations[j], as in the example above. In that case, annotatedCodeLocation.annotations[i].locations[j].fileLocation is considered to have the same value as annotatedCodeLocation.physicalLocation.fileLocation.

### properties property

An annotatedCodeLocation object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include additional information about the use of the location in this context that is not explicitly specified in the SARIF format.

## annotation object

### General

An annotation object associates a message with one or more physical locations.

### message property

An annotation object **SHALL** contain a property named message whose value is a message object (§3.9) that describes the physical location or locations specified by the locations property (§3.28.3).

### locations property

An annotation object **SHALL** contain a property named locations whose value is an array containing one or more unique (§3.6.2) physicalLocation objects (§3.21) to which the message (§3.28.2) is relevant.

## resources object

### General

A resources object represents items that can be localized, such as message strings and rule metadata.

### messageStrings property

A resources object **MAY** contain a property named messageStrings whose value is a JSON object (§3.5), each of whose properties represents a single localized string. The property names correspond to resource identifiers (§3.9.6) within message objects (§3.9). If the property name is used as the value of the messageId property (§3.9.9) of any message object in the current run, the property value **SHALL** be a plain text string (§3.9.2). If the property name is used as the value of the richMessageId property (§3.9.10) of any message object tin the current run, the property value **SHALL** be a rich text string (§3.9.3). A given resource identifier **SHALL NOT** appear both as the value of a messageId property and the value of a richMessageId property in the same run.

EXAMPLE:

"resources": {

"messageStrings": {

"call": "Function call",

"return": "Function return"

}

}

### rules property

A resources object **MAY** contain a property named rules whose value is a JSON object (§3.5), each of whose properties represents a rule object (§3.30).

If there is only one rule object with a particular id (§3.30.3), then the property name for that rule object **SHALL** be the rule id.

EXAMPLE 1: In this example, two rules have different ids. The property names match the rule ids.

"resources": {

"rules": {

"CA1001": {

"id": "CA1001",

"shortDescription": {

"text": "Types that own disposable fields should be

disposable."

}

},

"CA1002": {

"id": "CA1002",

"shortDescription": {

"text": "Do not expose generic lists."

}

}

}

}

Some tools use the same rule id to refer to multiple distinct (although logically related) rules. In that case, the property names for those rule objects **SHALL** be distinct, even though the rule ids are the same. The property names **SHOULD** be clearly related to the rule id.

EXAMPLE 2: In this example, two distinct but related rules have the same rule id. The property names are distinct and are clearly related to the rule id.

"resources": {

"rules": {

"CA1711-1": {

"id": "CA1711",

"messageStrings": {

"default": "Rename type name {0} so that it does not end in '{1}'"

}

},

"CA1711-2": {

"id": "CA1711",

"messageStrings": {

"default": "Either replace the suffix '{0}' in member name '{1}' with

the suggested numeric alternate or provide

a more meaningful suffix"

}

}

}

}

NOTE: This property is a dictionary, rather than simply an array of rule objects, to facilitate looking up the rule associated with each result object (§3.18) by means of the result's ruleId property (§3.18.4) or ruleKey property (§3.18.5).

## rule object

### General

A rule object contains information that describes a rule. We refer to this information as “rule metadata.”

### Constraints

Either the shortDescription property (§3.30.5) or the fullDescription property (§3.30.6) or both **SHALL** be present.

### id property

A rule object **SHALL** contain a property named id whose value is a string containing a stable, opaque identifier for the rule.

EXAMPLE: "CA2101"

NOTE: Rule identifiers must be stable for two reasons:

* So build automation scripts can refer to specific checks, for example, to disable them, without the risk of a script breaking if a rule id changes.
* So result management systems can compare results from one run to the next, without erroneously designating results as “new” because a rule id has changed.

Rule identifiers should be opaque – that is, they should not convey information to a user – because a rule's implementation might change over time. Suppose a rule id is "DoNotDoXOrY", suppose circumstances change so that “Y” is now acceptable, and suppose the implementation of the rule changes accordingly. Because the rule id must not change, the string "DoNotDoXOrY" will continue to be persisted to logs, where it will convey outdated guidance to users in a way that an opaque identifier such as "CA2101" would not.

### name property

A rule object **MAY** contain a property named name whose value is a message object (§3.9) containing a rule identifier that is understandable to an end user. If name contains implementation details that change over time, a tool author might alter a rule's name (while leaving the stable id property unchanged).

NOTE 1: A rule name is suitable in contexts where a readable identifier is preferable and where the lack of stability is not a concern.

NOTE 2: The name property is represented as a message object rather than as a string because it is intended to be understandable to an end user, so tool vendors might want to localize it.

EXAMPLE:

{ # A rule object

"name": {

"text": "SpecifyMarshalingForPInvokeStringArguments"

}

}

### shortDescription property

A rule object **MAY** contain a property named shortDescription whose value is a message object (§3.9) that provides a concise description of the rule. The shortDescription property **SHOULD** be a single sentence that is understandable when visible space is limited to a single line of text.

EXAMPLE:

{ # A rule object

"shortDescription": {

"text": "Specify marshaling for P/Invoke string arguments"

}

}

### fullDescription property

A rule object **SHOULD** contain a property named fullDescription whose value is a message object (§3.9) that describes the rule.

The fullDescription property **SHOULD**, as far as possible, provide details sufficient to enable resolution of any problem indicated by the result.

The first sentence of fullDescription **SHOULD** provide a concise description of the rule, suitable for display in cases where available space is limited. Tools that construct fullDescription in this way do not need to provide a value for shortDescription (§3.30.5). Tools that do not construct fullDescription in this way **SHOULD** provide a value for shortDescription, because otherwise, the initial portion of fullDescription that a viewer displays where available space is limited might not be understandable.

### messageStrings property

A rule object **MAY** contain a property named messageStrings whose value is a JSON object (§3.5) consisting of a set of properties with arbitrary names.

The value of each property **SHALL** be a plain text message string (§3.9.2). As with any message string, it **MAY** contain placeholders (§3.9.4) and embedded links (§3.9.5).

The set of property names appearing in the messageStrings property **SHALL** contain at least the set of strings which occur as values of result.ruleMessageId properties (§3.18.8) in the run. The messageStrings property **MAY** contain additional properties whose names do not appear as the value of the result.ruleMessageId property for any result in the run.

NOTE: Additional properties are permitted in the messageStrings property for the convenience of tool vendors, who might find it easier to emit the entire set of messages supported by a rule, rather than restricting it to those messages that happen to appear in the log file.

EXAMPLE:

{ # A rule object

"messageStrings": {

"objectCreation": "{0} creates a new instance of {1} which is never used.

Pass the instance as an argument to another method,

assign the instance to a variable,

or remove the object creation if it is unnecessary.",

"stringReturnValue": "{0} calls {1} but does not use the new string

instance that the method returns.

Pass the instance as an argument to another method,

assign the instance to a variable,

or remove the call if it is unnecessary."

}

}

### richMessageStrings property

If a rule object contains a messageStrings property (§3.30.7), it **MAY** also contain a property named richMessageStrings whose value is a JSON object (§3.5) consisting of a set of properties with arbitrary names.

The value of each property **SHALL** be a rich text message string (§3.9.3). As with any message string, it **MAY** contain placeholders (§3.9.4) and embedded links (§3.9.5).

The rules governing the set of property names appearing in the richMessageStrings property are the same as those for the messageStrings property.

SARIF consumers that cannot render rich text **SHALL** ignore the richMessageStrings property and use the messageStrings property instead. For this reason, every property name that appears in the richMessageStrings property **SHALL** also appear in the messageStrings property. SARIF consumers that can render rich text **SHOULD** use the richMessageStrings property, assuming they take appropriate measures to address security issues such as those discussed in §3.9.3.2.

### helpUri property

A rule object **MAY** contain a property named helpUri whose value is a string containing the URI where the primary documentation for the rule can be found.

NOTE: The documentation might include examples, contact information for the rule authors, and links to additional information about the rule.

### help property

A rule object **MAY** contain a property named help whose value is a message object (§3.9) which provides the primary documentation for the rule.

NOTE: This property is useful when help information is not available at a URI, for example, when the rule is a custom rule written by a developer, as opposed to one supplied by the tool vendor.

### configuration property

A rule object **MAY** contain a property named configuration whose value is a ruleConfiguration object (§3.31).

If this property is absent, it **SHALL** be taken to be present, and its properties **SHALL** be taken to have the default values specified in §3.31.

### properties property

A rule object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the rule that is not explicitly specified in the SARIF format.

This property **SHALL NOT** be used to hold rule configuration information. Use the ruleConfiguration.parameters property (§3.31.4) for that.

## ruleConfiguration object

### General

A ruleConfiguration object contains rule configuration information, that is, information about the rule that a SARIF producer can modify at runtime, before executing its scan. For example, if the rule specifies a maximum source file line length, its configuration information might specify the maximum permitted line length.

For an example, see §3.31.4.

### enabled property

A ruleConfiguration object **MAY** contain a property named enabled whose value is a Boolean that specifies whether the rule will be evaluated during the scan.

If this property is absent, its value **SHALL** be taken to be true.

EXAMPLE: In this example, a tool allows the user to enable or disable rules:

SecurityScanner --disable "SEC4002,SEC4003" --enable SEC6012

### defaultLevel property

A ruleConfiguration object **MAY** contain a property named defaultLevel whose value is one of the strings "warning", "error", "note", or "open", with the same meanings as when those strings appear as the value of the result.level property (§3.18.6).

If this property is absent, it **SHALL** be taken to have the value "warning".

The value of this property **SHALL** provide the value for the level property for any result object (§3.18) which refers to this rule configuration through its ruleId property (§3.18.3) or its ruleKey property (§3.18.5), and which does not itself specify a level property.

EXAMPLE: In this example, a tool allows the user to override a rule’s default level:

WebScanner --level "WEB1002:error,WEB1005:warning"

### parameters property

A ruleConfiguration object **MAY** contain a property named parameters whose value is a property bag (§3.7). This allows a rule to define configuration information that is specific to that rule.

EXAMPLE: In this example, a rule that specifies the maximum permitted source line length is parameterized by the value of the maximum length.

{ # A rule object (§3.30.)

"id": "SA2707",

"name": {

"text": "LimitSourceLineLength"

},

"shortDescription": {

"text": "Limit source line length for readability."

},

"configuration": {

"enabled": true,

"defaultLevel": "warning",

"parameters": {

"maxLength": 120

}

}

}

The rule provides a default value, but the tool allows the user to override it:

StyleScanner \*.c --rule-config "SA2707:maxLength=80"

## fix object

### General

A fix object represents a proposed fix for the problem indicated by the result object (§3.18) in which it occurs. It specifies a set of files to modify. For each file, it specifies regions to remove, and provides new file content to insert.

EXAMPLE:

{ # A result object (§3.18).

"fix": {

"description": { # See §3.32.2.

"text": "Private member names begin with '\_'"

},

"fileChanges": [ # See §3.32.3.

{ # A fileChange object (§3.33).

...

}

]

}

}

### description property

A fix object **SHOULD** contain a property named description whose value is a message object (§3.9) that describes the proposed fix.

NOTE: The purpose of the description property is to enable a SARIF viewer to present the proposed fix to the end user.

EXAMPLE:

"fix": {

"description": {

"text": "Combine declaration and initialization of variable 'x'."

},

...

}

### fileChanges property

A fix object **SHALL** contain a property named fileChanges whose value is an array of one or more fileChange objects (§3.33).

## fileChange object

### General

A fileChange object represents a change to a single file.

EXAMPLE:

{ # A fix object (§3.31).

"fileChanges": [ # See §3.32.3.

{

"fileLocation": { # See §3.33.2.

"uri": "a.h"

},

"replacements": [ # See §3.33.3.

{ # A replacement object (§3.34).

...

},

{ # Another replacement object.

...

}

]

}

]

}

### fileLocation property

A fileChange object **SHALL** contain a property named fileLocation whose value is a fileLocation object (§3.3) that represents the location of the file.

### replacements property

A fileChange object **SHALL** contain a property named replacements whose value is an array of one or more replacement objects (§3.34), each of which represents the replacement of a single region of the file specified by the fileLocation property (§3.33.2).

## replacement object

### General

A replacement object represents the replacement of a single region of a file. If the region’s length is zero, it represents an insertion point.

If a replacement object specifies both the removal of a region by means of the deletedRegion property (§3.34.3) and the insertion of new file content by means of the insertedContent property (§3.34.4), then the effect of the replacement **SHALL** be as if the removal were performed before the insertion.

If a single fileChange object (§3.33) specifies more than one replacement, then the effect of the replacements **SHALL** be as if they were performed in the order they appear in the replacements array (§3.33.3). The deletedRegion property of each replacement object **SHALL** specify the location of the replacement in the unmodified file.

EXAMPLE 1: Suppose a fileChange object contains a replacements property whose value is the following array of replacement objects:

"fileChanges": [

{

"deletedRegion": {

"offset": 12,

"length": 5

},

"insertedContent": {

"binary": "ZXhhbXBsZQ=="

}

},

{

"deletedRegion": {

"offset": 20,

"length": 3

}

},

{

"deletedRegion": {

"offset": 312,

"length": 0

},

"insertedContent": {

"binary": "ZXhhbXBsZQ=="

}

}

]

The first replacement object removes 5 bytes starting at offset 12; that is, it removes bytes 12–16. Then it inserts the 7 bytes specified by the MIME Base64-encoded string in the insertedContent.binary property at the same offset.

The second replacement object removes 3 bytes starting at offset 20 *with respect to the unmodified file*. Since 5 bytes were removed and 7 bytes inserted *before* byte 20, the 3 bytes removed actually start at byte 22. Since the insertedContent property is absent, no content is inserted in place of the deleted bytes.

In the third replacement object, the length of the region specified by the deletedRegion property is zero, so the region represents an insertion point. The 7 bytes specified by the insertedContent.binary property are inserted at offset 312 with respect to the unmodified file.

A replacement object can represent either a textual replacement or a binary replacement, depending on whether the deletedRegion property (§3.34.3) specifies a text region (§3.22.2) or a binary region (§3.22.3).

EXAMPLE 2: In this example, the replacements property specifies a replacement in a text file.

"replacements": [

{

"deletedRegion": { # The region object represents a text region ((§3.22.2)

"startLine": 12,

"startColumn": 5,

"endColumn": 9

},

"insertedContent": {

"text": "example" # The insertedContent property contains a text property

} # instead of a binary property.

}

]

When performing a replacement in a text file, the SARIF producer **SHOULD** specify a text replacement rather than a binary replacement. This allows the SARIF producer to specify the region without regard to whether the file starts with a byte order mark (BOM).

### Constraints

If the deletedRegion property (§3.34.3) specifies a text region (§3.22.2) and the insertedContent property (§3.34.4) is present, then the insertedContent property **SHOULD** contain a text property (§3.2.2).

If the deletedRegion property specifies a binary region (§3.22.3) and the insertedContent property is present, then the insertedContent property **SHALL** contain a binary property (§3.2.3).

### deletedRegion property

A replacement object **SHALL** contain a property named deletedRegion whose value is a region object (§3.22) specifying the region to delete.

If the length of the region specified by deletedRegion is zero, then deletedRegion specifies an insertion point, and the SARIF consumer performing the replacement **SHALL NOT** remove any file content.

### insertedContent property

A replacement object **MAY** contain a property named insertedContent whose value is a fileContent object (§3.2) that specifies the content to insert in place of the region specified by the deletedRegion property (or at the point specified by deletedRegion, if deletedRegion has a length of zero and therefore specifies an insertion point).

If insertedContent is absent or its properties specify content whose length is zero, the SARIF consumer performing the replacement **SHALL NOT** insert any content.

## notification object

### General

A notification object describes a condition encountered in the course of running an analysis tool which is relevant to the operation of the tool itself, as opposed to being relevant to a file being analyzed by the tool. Conditions relevant to files being analyzed by a tool are represented by result objects (§3.18).

### id property

A notification object **MAY** contain a property named id whose value is a string containing an identifier for the condition that was encountered.

NOTE: In contrast to rule identifiers (see rule.id, §3.30.3), which must be stable and opaque, notification identifiers do need to be either stable or opaque, because the reasoning that leads to those requirements for rule ids does not apply to tool notifications. A tool notification with level "error" should always be treated as a failure, and tools should not allow them to be disabled. And tool authors are free to change the notification ids at any time, so there is no reason for them to be opaque; to the contrary, they are more useful if they convey information to the user.

### ruleId property

If the condition described by the notification object is relevant to a particular analysis rule, the notification object **SHOULD** contain a property named ruleId whose value is a string containing the stable, unique identifier of the rule (§3.30.3).

### ruleKey property

If there is more than one rule with the id specified by the ruleId property (§3.35.3), and if the run object (§3.11) in which this notification object occurs contains a resources.rules property (§3.11.14, §3.29.3), then the notification object **SHALL** contain a property named ruleKey whose value is a string that matches one of the property names in the run.resources.rules object.

The value of the ruleId property on this notification object **SHALL** match the id property (§3.30.3) of the rule object (§3.29) identified by ruleKey.

EXAMPLE: In this example, there is more than one rule with id CA1711. When the log file includes a notification with that rule id, it also provides a value for ruleKey to specify which of the rules with that id is meant.

{ # A run object (§3.11).

"invocations": [ # See §3.11.8.

{ # An invocation object (§3.13).

"configurationNotifications": [ # See §3.13.21.

{ # A notification object.

"id": "CFG0001",

"message": {

"text": "Rule configuration is missing."

},

"ruleKey": "CA1711-1", # Specifies a property name within "rules".

"ruleId": "CA1711" # Matches the "id" value of the specified

# property value within "rules"

}

],

}

],

"resources": { # See §3.11.14.

"rules": { # See §3.29.3.

"CA1711-1": { # A rule object (§3.30).

"id": "CA1711",

...

},

"CA1711-2": { # Another rule object with the same id.

"id": "CA1711",

...

}

}

}

}

### physicalLocation property

If the condition described by the notification object is relevant to a particular file location, the notification object **SHOULD** contain a property named physicalLocation whose value is a physicalLocation object (§3.21) that identifies the relevant location.

### message property

A notification object **SHALL** contain a property named message whose value is a message object (§3.9) that describes the condition that was encountered.

NOTE: The message object in the notification.message property will typically not contain a richText (§3.9.8) or richMessageId (§3.9.10) property because tool notifications typically appear on the console, where rich text is not supported.

### level property

A notification object **MAY** contain a property named level whose value is one of a fixed set of strings that specify the severity level of the notification.

If present, the level property **SHALL** have one of the following values, with the specified meanings:

* "error": A serious problem was found. The condition encountered by the tool resulted in the analysis being halted, or caused the results to be incorrect or incomplete.
* "warning": A problem that is not considered serious was found. The condition encountered by the tool is such that it is uncertain whether a problem occurred, or is such that the analysis might be incomplete but the results that were generated are probably valid.
* "note": The notification is purely informational. There is no required action.

passlevel property is absent, it **SHALL** be considered equivalent to the value "warning".

### threadId property

A notification object **MAY** contain a property named threadId whose value is an integer which identifies the thread associated with this notification.

### time property

A notification object **MAY** contain a property named time whose value is a string specifying the date and time at which the analysis tool generated the notification. The string **SHALL** be in the format specified by (§3.8).

### exception property

If the notification is a result of a runtime exception, the notification object **MAY** contain a property named exception whose value is an exception object (§3.36).

If the notification is not the result of a runtime exception, the exception property **SHALL** be absent.

### properties property

A notification object **MAY** contain a property named properties whose value is a property bag (§3.7). This allows tools to include information about the encountered condition that is not explicitly specified in the SARIF format.

## exception object

### General

An exception object describes a runtime exception encountered in the course of executing an analysis tool. This includes signals in POSIX-conforming operating systems

### kind property

An exception object **SHOULD** contain a property named kind whose value is a string describing the exception.

If the exception represents a thrown object, kind **SHALL** be the fully qualified type name of the object that was thrown, if that information is available.

EXAMPLE 1: C#: "System.ArgumentNullException"

If the exception represents a POSIX signal, kind **SHALL** be the symbolic name of the signal as specified in <signal.h>.

EXAMPLE 2: POSIX: "SIGFPE"

If the tool does not have access to information about the object that was thrown, the kind property **SHALL** be absent.

### message property

An exception object **SHOULD** contain a property named message whose value is a string containing a plain text message string (§3.9.2) that describes the exception.

If the tool does not have access to an appropriate property of the thrown object, the message property **SHALL** be absent.

EXAMPLE 1: C++: The tool would populate message from the string returned from the what() method of any object derived from std::exception.

EXAMPLE 2: C#: The tool would populate message from the value of the Message property of any object derived from System.Exception.

NOTE: The exception.message property is not a message object (§3.9) because exception messages, appearing as they do in typical languages and operating systems, are inherently plain text, and require no arguments (§3.9.4).

### stack property

An exception object **MAY** contain a property named stack whose value is a stack object (§3.25) that describes the sequence of function calls leading to the exception.

### innerExceptions property

An exception object **MAY** contain a property named innerExceptions whose value is an array of one or more exception objects, each of which is considered a cause of the containing exception.

NOTE: There is commonly no more than one inner exception. This property is an array to accommodate platforms that provide a mechanism for aggregating exceptions, such as the System.AggregateException class from the .NET Framework.

# Conformance

## Conformance targets

This specification defines requirements for the SARIF file format and for certain software components that interact with it. The entities (“conformance targets”) for which this specification defines requirements are:

* **SARIF log file**
* **SARIF resource file**: A SARIF file that contains only those elements related to resources.
* **SARIF producer**: A program which emits output in the SARIF format.
* **Direct producer**: An analysis tool which acts as a SARIF producer.
* **Converter**: A SARIF producer that transforms the output of an analysis tool from its native output format into the SARIF format.
* **SARIF consumer**: A program that reads and interprets a SARIF log file.
* **Viewer**: A SARIF consumer that reads a SARIF log file, displays a list of the results it contains, and allows an end user to view each result in the context of the programming artifact in which it occurs.

The normative content in this specification defines requirements for SARIF log files, except for those normative requirements that are explicitly designated as defining the behavior of a SARIF producer, a direct producer, a converter, a SARIF consumer, or a viewer.

## Conformance Clause 1: SARIF log file

A text file satisfies the “SARIF log file” conformance profile if:

* It conforms to the syntax and semantics defined in §3.

## Conformance Clause 2: SARIF resource file

A text file satisfies the “SARIF resource file” conformance profile if:

* Its name conforms to the convention defined in §3.9.6.4, “SARIF resource file format”.
* It contains only those elements defined in §3.9.6.4.
* Those elements that it does contain conform to the syntax and semantics defined in §3, except as modified in §3.9.6.4.

## Conformance Clause 3: SARIF producer

A program satisfies the “SARIF producer” conformance profile if:

* It produces output in the SARIF format, according to the semantics defined in §3.
* It satisfies those normative requirements in §3 that are designated as applying to SARIF producers.

## Conformance Clause 4: Direct producer

An analysis tool satisfies the “Direct producer” conformance profile if:

* It satisfies the “SARIF producer” conformance profile.
* It additionally satisfies those normative requirements in §3 that are designated as applying to “direct producers” or to “analysis tools”.
* It does not emit any objects, properties, or values which, according to §3, are intended to be produced only by converters.

## Conformance Clause 5: Converter

A converter satisfies the “Converter” conformance profile if:

* It satisfies the “SARIF producer” conformance profile.
* It aditionally satisfies those normative requirements in §3 that are designated as applying to converters.
* It does not emit any objects, properties, or values which, according to §3, are intended to be produced only by direct producers.

## Conformance Clause 6: Deterministic producer

An analysis tool or a converter satisfies the “Deterministic producer” conformance profile if:

* It satisfies the “Direct producer” conformance profile or the “Converter” conformance profile, as appropriate.
* It satisfies the normative requirements in Appendix F, “Producing deterministic SARIF log files”.

## Conformance Clause 7: SARIF consumer

A consumer satisfies the “SARIF consumer” conformance profile if:

* It reads SARIF log files and interprets them according to the semantics defined in §3.
* It satisfies those normative requirements in §3 that are designated as applying to SARIF consumers.

## Conformance Clause 8: Viewer

A viewer satisfies the “viewer” conformance profile if:

* It satisfies the “SARIF consumer” conformance profile.
* It additionally satisfies the normative requirements in §3 that are designated as applying to viewers.

1. (Informative) Acknowledgments

(**Note:** A Work Product approved by the TC must include a list of people who participated in the development of the Work Product. This is generally done by collecting the list of names in this appendix. This list shall be initially compiled by the Chair, and any Member of the TC may add or remove their names from the list by request.

Remove this note before submitting for publication.)

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

[Participant Name, Affiliation | Individual Member]

[Participant Name, Affiliation | Individual Member]

1. (Informative) Use of fingerprints by result management systems

On large software projects, a single run of a set of analysis tools can produce hundreds of thousands of results or more. To deal with so many results, some software development teams adopt a strategy whereby they first prevent the introduction of new problems into their code, and then work to address the existing problems.

To prevent the introduction of new problems, it is necessary first to record the results from a designated run. We refer to this as a baseline. It is then necessary to compare the results from a subsequent run with the baseline.

To determine whether a result from a subsequent run is the same as a result from the baseline, there must be a way to use information contained in the result to construct a stable identifier for the result. We refer to this identifier as a fingerprint.

A result management system can construct a fingerprint by using information contained in the SARIF file such as

* the name of the tool that produced the result.
* the rule id.
* the file system path to the analysis target.

There are situations where information that would be helpful in uniquely identifying a result is not easily detectable by the result management system. For example, consider a tool which checks documentation for words that are culturally or politically sensitive. The word would most likely occur only in the message property, for example: "The word xxx should not be used in documentation."

The SARIF format provides the fngerprintContributions property to allow analysis tools and other components in the SARIF ecosystem to provide additional information which a result management system can incorporate into the fingerprint that it constructs for each result. In this example, the tool might set the value of a property in the fingerprintContributions object to the prohibited word.

Some information contained in the result is not useful in constructing a fingerprint. For example, suppose the fingerprint were to include the line number where the result was located, and suppose that after the baseline was constructed, a developer inserted additional lines of code above that location. Then in the next run, the result would occur on a different line, the computed fingerprint would change, and the result management system would erroneously report it as a new result.

It is difficult to devise an algorithm that constructs a truly stable fingerprint for a result. Fortunately, for practical purposes, the fingerprint does need to be absolutely stable; it only needs to be stable enough to reduce the number of results that are erroneously reported as “new” to a low enough level that the development team can manage the erroneously reported results without too much effort.

1. (Informative) Use of SARIF by log file viewers

It is frequently useful for an end user to view the results produced by an analysis tool in the context of the programming artifacts in which they occur. A log file viewer is a program that allows an end user to do this.

Typically, the user opens a log file in the viewer, which presents a list of the results in the log file. When the user selects a result from the list, the viewer displays the source code from the file specified in the result, and displays information about the result in the vicinity of the region where the result occurred. For example, the viewer might interleave result information between lines of source code.

There are various reasons why a viewer might need to know the type of information contained in a source file that it displays:

* If the viewer knows the programming language, it can provide services such as syntax highlighting.
* If the result occurs in a source file that is nested within (for example) a compressed container file, then the viewer needs to know the file type of the container so that it can extract the source file.

There are various ways that a viewer might obtain file type information. In the SARIF format, the mimeType property of the file object provides this information. In the absence of the mimeType property, a viewer can fall back to examining the filename extension, for example “.zip”. It is recommended that the analysis tool provide the mimeType property (which it must know, because it was able to interpret the file in which it detected the result), rather than forcing the viewer to rely on a file name extension.

1. (Informative) Production of SARIF by converters

There are two broad categories of tools that can produce output in the SARIF format. Analysis tools produce SARIF as a result of performing a scan on a set of analysis targets. Converters translate existing data from a non-SARIF format into the SARIF format. That data might come from an analysis tool that produces output in a non-SARIF format, from a bug database, or from any other source.

Converters should populate those elements of the SARIF format for which a direct equivalent exists in the input data.

If the input data includes information for which there is no SARIF equivalent, converters may use it to populate the various property bags and tag lists defined by the SARIF format, or they may simply omit it from the output. When populating a property bag with such information, converters should use a property name that matches the name of that piece of information in the native tool format, even if that name does not conform to the camelCase convention used in the rest of this specification. This makes it easier to match these properties with the source data in the native tool format.

NOTE: The converter must replace any characters that cannot occur in a JSON string with the appropriate escape sequence.

If the input data does not include an equivalent for any SARIF element, the converter should not attempt to synthesize that element. For example, a converter should not attempt to heuristically extract a rule id from the text of an unstructured error message.

If a converter were to synthesize values, it would potentially introduce additional complexity in the implementation of SARIF viewers. The reason is that the viewer itself might examine the analysis tool and its version in the tool object, and attempt to synthesize missing elements.

Now suppose a converter made a bad choice in synthesizing a missing element, and then fixed the problem in an update. As a result, two log files claiming to have been produced by the same version of the same analysis tools might have different elements filled in, or the same elements filled in differently. For that matter, two different converters might make different choices in how to synthesize missing elements. As a result, the viewer would have to take into account both the analysis tool (and its version) and the converter (and its version) in deciding how to synthesize any remaining elements.

By design, to avoid this added complexity, the SARIF standard does not define an element to hold the converter version. This, together with the guidance that converter implementers should not attempt to synthesize missing elements, allows viewer implementers to assume that all files from the same version of the same tool are identical in structure.

This general guidance is embodied in various sections of the specification. For example:

* A converter should not attempt to synthesize a ruleId for a result if the tool does not provide one.
* A converter that knows which file a result was detected in, but not which file the analysis tool was originally instructed to scan, should populate the location.resultFile property, but should not attempt to populate location.analysisTarget (see §3.20.2).
* A converter should not attempt to guess whether the analysis tool's version string is intended to be interpreted as a Semantic Version 2.0.0 version string (see §3.12.4).

1. (Informative) Locating rule metadata

The SARIF format allows rule metadata to be included in a SARIF log file (see §3.11.14 and §3.29). A SARIF log file does not need to include any rule metadata. This raises the questions of when rule metadata should be included in a log file, and how to locate the rule metadata if it is not included in the log file.

Rule metadata should be included in a log file in the following circumstances:

* The log file is intended to be viewed in a tool such as a log file viewer that needs to display rule metadata related to each result even when the tool is not connected to a network.
* The log file is intended to be uploaded to a result management system which requires information about every rule specified by every result, and which might not have prior knowledge of the rules specified by the results in this log file.
* Neither of the above applies, but the increased log file size due to the rule metadata is not considered significant.

If rule metadata is not included in the log file, this specification does not specify a mechanism for locating the metadata. If the SARIF log file is produced in the context of an engineering system that provides a service from which rule metadata can be obtained (for example, a result management system, or a web service dedicated to rule metadata), then tooling can be created to merge a log file with the relevant metadata when required (for example, when presenting the results in a log file viewer).

1. (Normative) Producing deterministic SARIF log files
   1. General

In certain circumstances, it is desirable for an analysis tool to produce deterministic output; that is, for it to produce identical output when run repeatedly over identical inputs.

Certain build systems provide an example of when this is desirable. Consider a build system that caches the results of each build step. If the build is rerun, and the inputs to the step are identical (which the build system might determine, for example, by comparing timestamps, or by computing a hash of the inputs to the step and storing it along with the output from the step), then the build system can save time by not re-running the step, and simply using the existing outputs.

In the case of SARIF, one could imagine a sequence of build steps where Steps A, B, and C each run an analysis tool on a different set of targets, producing log files A.sarif, B.sarif, and C.sarif, and then build Step D performs an analysis on the aggregate of those log files. If the targets analyzed in Step B change but the targets analyzed in steps A and C do not, and if the contents of the SARIF log file are deterministic, then when the build is re-run, only Steps B and D need to be performed.

Authors of analysis tools are encouraged to provide a mechanism (for example, a command line option such as --deterministic) which instructs the tool to produce deterministic output.

There are several issues to consider when producing deterministic output:

* Avoiding elements of the SARIF file format whose values are non-deterministic.
* Emitting array and dictionary elements in a deterministic order.
* Avoiding absolute paths.
* Handling baseline information
  1. Non-deterministic file format elements

A tool that produces deterministic output **SHALL NOT** emit the following elements of the SARIF format. All of these elements are **OPTIONAL**.

Not all of these elements are non-deterministic in all cases. For example, some build systems might run all builds on the same machine or under the same account. However, avoiding these elements, in conjunction with the techniques described in subsequent sections of this Appendix, guarantees deterministic output.

* invocation.startTime
* invocation.endTime
* invocation.processId
* invocation.machine
* invocation.account
* invocation.fileName (because fileName is specified as being an absolute path, and tools might be stored in different directories on different machines)
* invocation.workingDirectory
* invocation.environmentVariables
* The use of absolute file paths in invocation.commandLine (because builds performed on different machines might use a different root directory)
* annotatedCodeLocation.threadId
* notification.threadId
* notification.time
* run.id
* run.automationId
* run.baselineId
* run.originalUriBaseIds
* stackFrame.threadId
* stackFrame.address (because security measures such as address space layout randomization (ASLR) might place identical code at different addresses from run to run)
* The presence of any non-deterministic elements in a property bag property
  1. Array and dictionary element ordering

A tool that produces deterministic output **SHALL** emit array and dictionary elements in a deterministic order.

For some arrays, the SARIF format requires a specific ordering. For example, within the stack.Frames property, SARIF requires the annotatedCodeLocation object representing the most deeply nested function call to appear first.

For other arrays, the SARIF format does not require a specific ordering. For example, within the file.hashes property, SARIF does not require the hash objects to appear in any particular order. For such arrays, a tool can ensure the order by sorting the array elements before writing them to the log file. For example, it might sort the hash objects alphabetically by the string value of the hash.algorithm property.

A tool might similarly choose to emit the string elements of a properties.tags array in locale-insensitive alphabetical order.

The array of result objects in the run.results array presents more of a problem. A multi-threaded analysis tool analyzing multiple files in parallel might produce results in any order, and there is no natural order for the results. A tool might choose to order them, for example, first alphabetically by analysis target URI, then numerically by line number, then by column number, then alphabetically by rule id.

For dictionaries such as the run.rules object or the run.files object, a tool might order the property names alphabetically, using a locale-insensitive ordering.

* 1. Absolute paths

The use of absolute file paths in fileLocation.uri properties makes it difficult to produce deterministic output. For example:

* Different build machines might be configured to use different source directories.
* A single build machine might use a different directory for each build.

A tool that produces deterministic output **SHALL NOT** emit absolute file paths. Tools can achieve this by emitting URIs that are relative to one or more root directories (for example, a source root directory and an output root directory), and accompanying each fileLocation.uri property with the corresponding fileLocation.uriBaseId property.

* 1. Compensating for non-deterministic output

If an analysis tool does not produce deterministic output, a build system can add additional processing steps to compensate.

There are two scenarios to consider:

* Log equality is determined by a simple comparison of file contents, or by comparing file hashes.
* Log equality is determined by an “intelligent” comparison.

In the first scenario, a post-processing step could produce deterministic output by creating a new file that omits non-deterministic elements, reorders array elements and object properties, removes file path prefixes, and introduces fileLocation.uriBaseId properties.

In the second scenario, a post-processing step could intelligently compare the newly produced log to the log from a previous build by ignoring non-deterministic elements, ensuring that arrays have the same elements regardless of order, and ignoring file path prefixes.

* 1. Interaction between determinism and baselining

SARIF's baselining feature poses a particular challenge for determinism. We illustrate the problem with the following scenario:

On a particular date, a project's nightly build runs an analysis tool ToolX, which produces a log file, say, log\_20170914.sarif. The next day, a developer modifies one of the files scanned by the tool in a way that introduces a new problem. That night, the nightly build tool runs again, this time producing a log file which compares the current set of results to those that appeared in the previous run:

ToolX --input a.c b.c --baseline log\_20170914.sarif --output log\_20170915.sarif

Because a new problem has been introduced, log\_20170614.sarif will contain a result object whose baselineState is "new". The next night, without any further changes to the source files, the tool is run yet again:

ToolX --input a.c b.c --baseline log\_20170915.sarif --output log\_20170916.sarif

The result object that first appeared in log\_20160615.sarif still appears in log\_20160616.sarif, but since it existed in the baseline, its baselineState will now be "existing".

The result is that even though none of the analysis target files have changed, the log file has changed, or at least, a simple file comparison (such as comparing the hash of the new log with the hash of the baseline) will report that is has changed.

Strictly speaking, this does not violate determinism. After all, the baseline file has changed, and the baseline file is one of the inputs to the analysis. But from a practical standpoint, this is still a problem, albeit a small one.

If the build uses a simple mechanism such as hash value comparison to determine if a file has changed, then on those occasions when the only difference between the newest log and the baseline is that some results that were previously "new" are now "existing", subsequent build steps which consume the SARIF log file will run, even if they might not actually be necessary. For example, a build step which automatically files bugs for new results will run, even though the log contains no new results. Or a build step which tracks the number of open issues will run, even though the number of open issues has not actually changed.

If the build engineers for a project wish to absolutely minimize the execution of unnecessary build steps, they have various options. They might perform an “intelligent” comparison between the baseline and the new log, treating "new" results in the baseline as equivalent to "existing" results. Or they might rewrite the baseline (marking all "new" results as "existing") before performing the comparison. Of course, there is no guarantee that such an “intelligent” comparison or baseline rewriting process will actually take less time than the unnecessary build steps it is intended to avoid.

1. (Informative) Guidance on fixes

Tools that produce SARIF files which include fix objects should take care to structure those fixes in such a way as to affect a minimal range of bytes. This maximizes the likelihood that an automated tool can safely apply multiple fixes to the same file.

The following example will clarify what this means and why it is important. Consider an XML file containing the following element:

<lineItem partNumber=A3101 />

Suppose that a (domain-specific) XML scanning tool reported two results:

* The value of the partNumber attribute is not enclosed in quotes.
* The part numbering scheme has changed, and part numbers beginning with “A” now begin with “AA”.

Fixing only result #1 would produce the element

<lineItem partNumber="A3101" />

Fixing only result #2 would produce the element

<lineItem partNumber=AA3101 />

Fixing both results should produce the element

<lineItem partNumber="AA3101" />

The fix for result #1 might be specified in various ways, for example:

1. As a single replacement:
   * Replace the characters A3101 with the characters "A3101".
2. As a sequence of two replacements:
   1. Insert a quotation mark before A3101.
   2. Insert a quotation mark after A3101.

The fix for result #2 is most simply specified as a single replacement:

* Replace the characters A3101 with the characters AA3101.

Suppose there exists an automated tool which reads a SARIF file containing fix objects and applies as many of the specified fixes as possible to the source files.

If the fix for result #1 were structured as a single replacement, then after applying the fix, the tool would not be able to fix result #2, because the range of characters specified by the fix for result #2 would have been replaced. On the other hand, if the fix for result #1 were structured as two replacements (with a separate insertion for each quotation mark), the tool would still be able to apply the fix for result #2, because the targeted range of characters would still exist.

Therefore, structuring fixes as sequences of minimal, disjoint byte range replacements maximizes the amount of work that can be done by automated fixup tools.

1. (Informative) Examples

This Appendix contains examples of complete, valid SARIF files, to complement the fragments shown in examples throughout this document.

* 1. Minimal valid SARIF log file

This is a minimal valid SARIF log file. It contains only those elements required by the specification (that is, those elements which the specification states **SHALL** be present).

The file contains a single run object (§3.11) with an empty results array (§3.11.13), as would happen if the tool detected no issues in any of the files it scanned.

{

"version": "2.0.0",

"runs": [

{

"tool": {

"name": "CodeScanner",

"semanticVersion": "2.1.0"

},

"results": [

]

}

]

}

* 1. Minimal recommended SARIF log file with source information

This is a minimal recommended SARIF log file for the case where

1. The analysis tool was run with the intent of scanning files and producing results (see §3.11.13), and
2. The analysis tool has source location information available.

The file contains those elements recommended by the specification (that is, those elements which the specification states “**SHOULD**” be present), in addition to the required elements.

The file contains a single run object (§3.11) with a results array (§3.11.13). The results array contains a single result object (§3.18) so the recommended elements of the result object can be shown.

It contains a run.files property (§3.11.10) specifying only those files in which the tool detected a result.

It does not contain a run.logicalLocations property (§3.11.12), because when physical location information is available, that property is optional (it “**MAY**” be present).

This example also includes a run.rules property (§3.11.14) containing rule metadata, even though rule metadata is optional, to show how a SARIF log file can be self-contained, in the sense of containing all the information necessary to interpret the results.

{

"version": "2.0.0",

"runs": [

{

"tool": {

"name": "CodeScanner",

"semanticVersion": "2.1.0"

},

"files": {

"file:///user/builder/work/src/collections/list.cpp": {

"mimeType": "text/x-c"

}

},

"results": [

{

"ruleId": "C2001",

"message": {

"text": "Variable \"count\" was used without being initialized.",

"richText": "Variable `count` was used without being initialized."

},

"locations": [

{

"analysisTarget": {

"uri": "file:///user/builder/work/src/collections/list.cpp",

"region": {

"startLine": 15

}

},

"fullyQualifiedLogicalName": "collections::list:add"

}

]

}

],

"rules": {

"C2001": {

"id": "C2001",

"fullDescription": {

"text": "A variable was used without being initialized. This can result in

runtime errors such as null reference exceptions."

}

}

}

}

]

}

* 1. Minimal recommended SARIF log file without source information

This is a minimal recommended SARIF file for the case where

1. The analysis tool was run with the intent of scanning files and producing results (see §3.11.13), but
2. The analysis tool does not have source location information available.

The file contains those elements recommended by the specification (that is, those elements which the specification states “**SHOULD**” be present), in addition to the required elements.

The file contains a single run object (§3.11) with a results array (§3.11.13). The results array contains a single result object (§3.18) so the recommended elements of the result object can be shown.

It contains a run.files property (§3.11.10) specifying only those files in which the tool detected a result.

It contains a run.logicalLocations property (§3.11.12), because when physical location information is not available, that property is recommended (it “**SHOULD**” be present).

{

"version": "2.0.0",

"runs": [

{

"tool": {

"name": "BinaryScanner",

"semanticVersion": "1.0.1"

},

"files": {

"file:///user/builder/work/bin/example": {

"mimeType": "application/vnd.microsoft.portable-executable"

}

},

"logicalLocations": {

"Example": {

"name": "Example",

"kind": "namespace"

},

"Example.Worker": {

"name": "Worker",

"kind": "type",

"parentKey": "Example"

},

"Example.Worker.DoWork": {

"name": "DoWork",

"kind": "function",

"parentKey": "Example.Worker"

}

},

"results": [

{

"ruleId": "B6412",

"message": {

"text": "The insecure method \"Crypto.Sha1.Encrypt\" should not be used.",

"richText": "The insecure method `Crypto.Sha1.Encrypt` should not be used."

},

"level": "warning",

"locations": [

{

"fullyQualifiedLogicalName": "Example.Worker.DoWork"

}

]

}

]

}

]

}

* 1. SARIF resource file with rule metadata

This sample demonstrates the use of SARIF for exporting a tool's rule metadata. The file conforms to the SARIF resource file format (§3.9.6.4) and contains rule metadata for the language specified by tool.language (§3.12.7).

{

"version": "2.0.0",

"runs": [

{

"tool": {

"name": "BinaryAnalyzer",

"semanticVersion": "2.1.0",

"language": "en-US"

},

"resources": {

"rules": {

"BA2006": {

"id": "BA2006",

"name": {

"text": "BuildWithSecureTools"

},

"shortDescription": {

"text": "Application code should be compiled with

the most up-to-date tool sets."

},

"fullDescription": {

"text": "Application code should be compiled with

the most up-to-date tool sets. The latest

version is 2.2."

},

"messageStrings": {

"Error\_BadModule": "built with {0} compiler version {1}

(Front end version {2})",

"Pass": "{0} was built with tools that satisfy

configured policy.",

"Error": "{0} was compiled with one or tools that do not

satisfy configured policy.",

"NotApplicable\_InvalidMetadata": "{0} was not evaluated for

check '{1}'."

},

"defaultLevel": "warning",

"helpUri": "http://www.example.com/tools/BinaryAnalyzer/rules/BA2006"

}

}

}

}

]

}

* 1. Comprehensive SARIF file

The purpose of this example is to demonstrate the usage of as many SARIF elements as possible. Not all elements are shown, because some are mutually exclusive.

Because the purpose is to present as many elements as possibly, the file as a whole does not represent best practices for SARIF usage, nor does it represent the output of a single, coherent analysis. For example, the result presented in the file involves a runtime exception, but at the same time it is marked as suppressedExternally (to demonstrate the result.suppressionStates property), which is unrealistic.

{

"version": "2.0.0",

"$schema": "http://json.schemastore.org/sarif-2.0.0",

"runs": [

{

"id": "BC650830-A9FE-44CB-8818-AD6C387279A0",

"stableId": "Nightly code scan",

"baselineId": "0A106451-C9B1-4309-A7EE-06988B95F723",

"automationId": "Build-14.0.1.2-Release-20160716-13:22:18",

"architecture": "x86",

"tool": {

"name": "CodeScanner",

"fullName": "CodeScanner 1.1 for Unix (en-US)",

"version": "2.1",

"semanticVersion": "2.1.0",

"fileVersion": "2.1.0.0",

"language": "en-US",

"sarifLoggerVersion": "1.25.0",

"properties": {

"copyright": "Copyright (c) 2017 by Example Corporation.

All rights reserved."

}

},

"invocations": [

{

"commandLine": "CodeScanner @collections.rsp",

"responseFiles": {

"collections.rsp": "-input src/collections/\*.cpp -log out/collections.sarif -rules all -disable C9999"

},

"startTime": "2016-07-16T14:18:25Z",

"endTime": "2016-07-16T14:19:01Z",

"machine": "BLD01",

"account": "buildAgent",

"processId": 1218,

"fileName": "/bin/tools/CodeScanner",

"workingDirectory": "/home/buildAgent/src",

"environmentVariables": {

"PATH": "/usr/local/bin:/bin:/bin/tools:/home/buildAgent/bin",

"HOME": "/home/buildAgent",

"TZ": "EST"

},

"configurationNotifications": [

{

"id": "UnknownRule",

"ruleId": "ABC0001",

"level": "warning",

"message": {

"text": "Could not disable rule \"ABC0001\" because

there is no rule with that id."

}

}

],

"toolNotifications": [

{

"id": "CTN0001",

"level": "note",

"message": {

"text": "Run started."

}

},

{

"id": "CTN9999",

"ruleId": "C2152",

"level": "error",

"message": {

"text": "Exception evaluating rule \"C2152\". Rule disabled;

run continues."

},

"physicalLocation": {

"fileLocation": {

"uri": "file:///home/buildAgent/src/crypto/hash.cpp"

}

},

"threadId": 52,

"time": "2016-07-16T14:18:43.119Z",

"exception": {

"kind": "ExecutionEngine.RuleFailureException",

"message": {

"text": "Unhandled exception during rule evaluation."

},

"stack": {

"frames": [

{

"message": {

"text": "Exception thrown"

},

"module": "RuleLibrary",

"threadId": 52,

"fullyQualifiedLogicalName":

"Rules.SecureHashAlgorithmRule.Evaluate",

"address": 10092852

},

{

"module": "ExecutionEngine",

"threadId": 52,

"fullyQualifiedLogicalName":

"ExecutionEngine.Engine.EvaluateRule",

"address": 10073356

}

]

},

"innerExceptions": [

{

"kind": "System.ArgumentException",

"message": "length is < 0"

}

]

}

},

{

"id": "CTN0002",

"level": "note",

"message": {

"text": "Run ended."

}

}

]

}

],

"files": {

"file:///home/buildAgent/src/collections/list.cpp": {

"mimeType": "text/x-c",

"length": 980,

"hashes": [

{

"algorithm": "sha256",

"value": "b13ce2678a8807ba0765ab94a0ecd394f869bc81"

}

]

},

"file:///home/buildAgent/bin/app.zip": {

"mimeType": "application/zip"

},

"file:///home/buildAgent/bin/app.zip#/docs/intro.docx": {

"uri": "/docs/intro.docx",

"mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",

"parentKey": "file:///home/buildAgent/bin/app.zip",

"offset": 17522,

"length": 4050

}

},

"logicalLocations": {

"collections::list::add": {

"name": "add",

"kind": "function",

"parentKey": "collections::list"

},

"collections::list": {

"name": "list",

"kind": "type",

"parentKey": "collections"

},

"collections": {

"name": "collections",

"kind": "namespace"

}

},

"results": [

{

"ruleId": "C2001",

"ruleMessageId": "default",

"message": {

"arguments": [

"ptr"

]

},

"suppressionStates": [ "suppressedExternally" ],

"baselineState": "existing",

"level": "error",

"snippet": {

"text": "add\_core(ptr, offset, val);\n return;"

},

"locations": [

{

"analysisTarget": {

"fileLocation": {

"uri": "file:///home/buildAgent/src/collections/list.cpp"

}

},

"resultFile": {

"fileLocation": {

"uri": "file:///home/buildAgent/src/collections/list.h"

},

"region": {

"startLine": 15,

"startColumn": 9,

"endLine": 15,

"endColumn": 10,

"length": 1,

"offset": 254

}

},

"fullyQualifiedLogicalName": "collections::list:add",

"decoratedName": "?add@list@collections@@QAEXH@Z"

}

],

"relatedLocations": [

{

"message": {

"text": "Variable \"ptr\" was declared here.",

"richText": "Variable `ptr` was declared here."

},

"physicalLocation": {

"fileLocation": {

"uri": "file:///home/buildAgent/src/collections/list.h"

},

"region": {

"startLine": 8,

"startColumn": 5

}

},

"fullyQualifiedLogicalName": "collections::list:add"

}

],

"codeFlows": [

{

"message": {

"text": "Path from declaration to usage"

},

"locations": [

{

"step": 1,

"kind": "declaration",

"target": "ptr",

"importance": "essential",

"message": {

"text": "Variable \"ptr\" declared.",

"richText": "Variable `ptr` declared."

},

"snippet": {

"text": "int \*ptr;"

},

"physicalLocation": {

"fileLocation": {

"uri": "file:///home/buildAgent/src/collections/list.h"

},

"region": {

"startLine": 15

}

},

"fullyQualifiedLogicalName": "collections::list:add",

"module": "platform",

"threadId": 52

},

{

"step": 2,

"kind": "assignment",

"target": "offset",

"values": [

"43"

],

"state": {

"y": "2",

"z": "4",

"y + z": "6",

"q": "7"

},

"importance": "unimportant",

"snippet": {

"text": "offset = (y + z) \* q + 1;"

},

"physicalLocation": {

"fileLocation": {

"uri": "file:///home/buildAgent/src/collections/list.h"

},

"region": {

"startLine": 15

}

},

"annotations": [

{

"message": {

"text": "(y + z) = 42",

"richText": "`(y + z) = 42`"

},

"locations": [

{

"region": {

"startLine": 15,

"startColumn": 13,

"endColumn": 19

}

}

]

}

],

"fullyQualifiedLogicalName": "collections::list:add",

"module": "platform",

"threadId": 52

},

{

"step": 3,

"kind": "call",

"importance": "essential",

"message": {

"text": "Uninitialized variable \"ptr\" passed to

method \"add\_core\".",

"richText": "Uninitialized variable `ptr` passed to

method `add\_core`."

},

"snippet": {

"text": "add\_core(ptr, offset, val)"

},

"target": "collections::list:add\_core",

"physicalLocation": {

"fileLocation": {

"uri": "file:///home/buildAgent/src/collections/list.h"

},

"region": {

"startLine": 25

}

},

"fullyQualifiedLogicalName": "collections::list:add",

"module": "platform",

"threadId": 52

}

]

}

],

"stacks": [

{

"message": {

"text": "Call stack resulting from usage of uninitialized variable."

},

"frames": [

{

"message": {

"text": "Exception thrown."

},

"physicalLocation": {

"fileLocation": {

"uri": "file:///home/buildAgent/src/collections/list.h"

},

"region": {

"startLine": 110,

"startColumn": 15

}

},

"module": "platform",

"threadId": 52,

"fullyQualifiedLogicalName": "collections::list:add\_core",

"address": 10092852,

"offset": 16,

"parameters": [ "null", "0", "14" ]

},

{

"physicalLocation": {

"fileLocation": {

"uri": "file:///home/buildAgent/src/collections/list.h"

},

"region": {

"startLine": 43,

"startColumn": 15

}

},

"module": "platform",

"threadId": 52,

"fullyQualifiedLogicalName": "collections::list:add",

"address": 10092176,

"offset": 84,

"parameters": [ "14" ]

},

{

"physicalLocation": {

"fileLocation": {

"uri": "file:///home/buildAgent/src/application/main.cpp"

},

"region": {

"startLine": 28,

"startColumn": 9

}

},

"module": "application",

"threadId": 52,

"fullyQualifiedLogicalName": "main",

"address": 10091200,

"offset": 156

}

]

}

],

"fixes": [

{

"description": {

"text": "Initialize the variable to null"

},

"fileChanges": [

{

"fileLocation": {

"uri": "file:///home/buildAgent/src/collections/list.h"

},

"replacements": [

{

"deletedRegion": {

"offset": 109

},

"insertedContent": {

"binary": "PSBudWxs"

}

}

]

}

]

}

]

}

],

"rules": {

"C2001": {

"id": "C2001",

"shortDescription": {

"text": "A variable was used without being initialized."

},

"fullDescription": {

"text": "A variable was used without being initialized. This can result

in runtime errors such as null reference exceptions."

},

"messageStrings": {

"default": "Variable \"{0}\" was used without being initialized."

},

"richMessageStrings": {

"default": "Variable `{0}` was used without being initialized."

}

}

}

}

]

}

1. (Informative) Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Editor** | **Changes Made** |
| 01 | 2017/09/22 | Laurence J. Golding | Initial version, transcribed from contribution with minor corrections. |
| 02 | 2017/11/29 | Laurence J. Golding | Incorporated changes for GitHub Issues [#25](https://github.com/oasis-tcs/sarif-spec/issues/25), [#27](https://github.com/oasis-tcs/sarif-spec/issues/27), and [#56](https://github.com/oasis-tcs/sarif-spec/issues/56). |
| 03 | 2018/01/10 | Laurence J. Golding | Incorporated changes for GitHub Issues [#33](https://github.com/oasis-tcs/sarif-spec/issues/33), #[61](https://github.com/oasis-tcs/sarif-spec/issues/61), [#69](https://github.com/oasis-tcs/sarif-spec/issues/69), and [#72](https://github.com/oasis-tcs/sarif-spec/issues/72). Made several minor editorial changes and a few changes to correct inaccuracies. |
| 04 | 2018/01/11 | Laurence J. Golding | Incorporated changes for GitHub Issue [#73](https://github.com/oasis-tcs/sarif-spec/issues/73). |
| 05 | 2018/01/15 | Laurence J. Golding | Incorporated changes for GitHub Issue [#79](https://github.com/oasis-tcs/sarif-spec/issues/79). |
| 06 | 2018/01/16 | Laurence J. Golding | Two minor editorial changes. |
| 07 | 2018/01/17 | Laurence J. Golding | Incorporated changes for GitHub Issue [#65](https://github.com/oasis-tcs/sarif-spec/issues/65). |
| 08 | 2018/02/19 | Laurence J. Golding | Incorporated changes for GitHub Issues [#66](https://github.com/oasis-tcs/sarif-spec/issues/66), [#74](https://github.com/oasis-tcs/sarif-spec/issues/74), [#81](https://github.com/oasis-tcs/sarif-spec/issues/81), #[88](https://github.com/oasis-tcs/sarif-spec/issues/88). |
| 09 | 2018/02/28 | Laurence J. Golding | Incorporate changes for GitHub Issues [#82](https://github.com/oasis-tcs/sarif-spec/issues/82), [#83](https://github.com/oasis-tcs/sarif-spec/issues/83), [#89](https://github.com/oasis-tcs/sarif-spec/issues/89), [#90](https://github.com/oasis-tcs/sarif-spec/issues/90), [#91](https://github.com/oasis-tcs/sarif-spec/issues/91), [#92](https://github.com/oasis-tcs/sarif-spec/issues/92), [#94](https://github.com/oasis-tcs/sarif-spec/issues/94), and [#104](https://github.com/oasis-tcs/sarif-spec/issues/104). |
| 10 | 2018/03/16 | Laurence J. Golding | Incorporate changes for GitHub Issues [#10](https://github.com/oasis-tcs/sarif-spec/issues/10), [#15](https://github.com/oasis-tcs/sarif-spec/issues/15), [#23](https://github.com/oasis-tcs/sarif-spec/issues/23), [#29](https://github.com/oasis-tcs/sarif-spec/issues/29), [#63](https://github.com/oasis-tcs/sarif-spec/issues/63), [#64](https://github.com/oasis-tcs/sarif-spec/issues/64), [#84](https://github.com/oasis-tcs/sarif-spec/issues/84), [#102](https://github.com/oasis-tcs/sarif-spec/issues/102), [#110](https://github.com/oasis-tcs/sarif-spec/issues/110). |