Static Analysis Results Interchange Format (SARIF) Version 2.0

Committee Specification Draft 02 (2.0.0-csd.2.beta.2019-02-20)

20 February 2019

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 ([v-lgold@microsoft.com](mailto:v-lgold@microsoft.com)), [Microsoft](http://www.microsoft.com)

Additional artifacts:

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

* JSON schemas:
  + sarif-schema.json
  + sarif-external-property-file-schema.json

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/v2.0/csd01/sarif-v2.0-csd01.docx

Permanent “Latest version” URI:  
http://docs.oasis-open.org/sarif/sarif/v2.0/sarif-v2.0.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 13](#_Toc1571707)

[1.1 IPR Policy 13](#_Toc1571708)

[1.2 Terminology 13](#_Toc1571709)

[1.3 Normative References 19](#_Toc1571710)

[1.4 Non-Normative References 20](#_Toc1571711)

[2 Conventions 21](#_Toc1571712)

[2.1 General 21](#_Toc1571713)

[2.2 Format examples 21](#_Toc1571714)

[2.3 Property notation 21](#_Toc1571715)

[2.4 Syntax notation 21](#_Toc1571716)

[3 File format 22](#_Toc1571717)

[3.1 General 22](#_Toc1571718)

[3.2 SARIF file naming convention 22](#_Toc1571719)

[3.3 fileContent objects 22](#_Toc1571720)

[3.3.1 General 22](#_Toc1571721)

[3.3.2 text property 22](#_Toc1571722)

[3.3.3 binary property 22](#_Toc1571723)

[3.4 fileLocation objects 23](#_Toc1571724)

[3.4.1 General 23](#_Toc1571725)

[3.4.2 Constraints 23](#_Toc1571726)

[3.4.3 uri property 23](#_Toc1571727)

[3.4.4 uriBaseId property 23](#_Toc1571728)

[3.4.5 fileIndex property 25](#_Toc1571729)

[3.4.6 Guidance on the use of fileLocation objects 26](#_Toc1571730)

[3.5 String properties 26](#_Toc1571731)

[3.5.1 General 26](#_Toc1571732)

[3.5.2 Redactable string properties 26](#_Toc1571733)

[3.5.3 GUID-valued string properties 27](#_Toc1571734)

[3.5.4 Hierarchical strings 27](#_Toc1571735)

[3.5.4.1 General 27](#_Toc1571736)

[3.5.4.2 Versioned hierarchical strings 27](#_Toc1571737)

[3.6 Object properties 28](#_Toc1571738)

[3.7 Array properties 28](#_Toc1571739)

[3.7.1 General 28](#_Toc1571740)

[3.7.2 Array properties with unique values 28](#_Toc1571741)

[3.8 Property bags 28](#_Toc1571742)

[3.8.1 General 28](#_Toc1571743)

[3.8.2 Tags 29](#_Toc1571744)

[3.8.2.1 General 29](#_Toc1571745)

[3.8.2.2 Tag metadata 29](#_Toc1571746)

[3.9 Date/time properties 30](#_Toc1571747)

[3.10 URI-valued properties 31](#_Toc1571748)

[3.10.1 General 31](#_Toc1571749)

[3.10.2 URIs that use the file scheme 32](#_Toc1571750)

[3.10.3 Internationalized Resource Identifiers (IRIs) 32](#_Toc1571751)

[3.11 message objects 32](#_Toc1571752)

[3.11.1 General 32](#_Toc1571753)

[3.11.2 Plain text messages 32](#_Toc1571754)

[3.11.3 Rich text messages 33](#_Toc1571755)

[3.11.3.1 General 33](#_Toc1571756)

[3.11.3.2 Security implications 33](#_Toc1571757)

[3.11.4 Messages with placeholders 33](#_Toc1571758)

[3.11.5 Messages with embedded links 34](#_Toc1571759)

[3.11.6 Message string resources 36](#_Toc1571760)

[3.11.6.1 General 36](#_Toc1571761)

[3.11.6.2 Embedded string resource lookup procedure 36](#_Toc1571762)

[3.11.6.3 SARIF resource file naming convention 37](#_Toc1571763)

[3.11.6.4 SARIF resource file lookup procedure 37](#_Toc1571764)

[3.11.6.5 SARIF resource file format 38](#_Toc1571765)

[3.11.6.5.1 General 38](#_Toc1571766)

[3.11.6.5.2 sarifLog object 38](#_Toc1571767)

[3.11.6.5.3 run object 38](#_Toc1571768)

[3.11.6.5.4 tool object 39](#_Toc1571769)

[3.11.6.5.5 resources object 39](#_Toc1571770)

[3.11.7 text property 39](#_Toc1571771)

[3.11.8 richText property 39](#_Toc1571772)

[3.11.9 messageId property 39](#_Toc1571773)

[3.11.10 richMessageId property 39](#_Toc1571774)

[3.11.11 arguments property 40](#_Toc1571775)

[3.12 sarifLog object 40](#_Toc1571776)

[3.12.1 General 40](#_Toc1571777)

[3.12.2 version property 40](#_Toc1571778)

[3.12.3 $schema property 41](#_Toc1571779)

[3.12.4 runs property 41](#_Toc1571780)

[3.13 run object 41](#_Toc1571781)

[3.13.1 General 41](#_Toc1571782)

[3.13.2 externalPropertyFiles property 41](#_Toc1571783)

[3.13.2.1 Rationale 41](#_Toc1571784)

[3.13.2.2 Property definition 42](#_Toc1571785)

[3.13.3 id property 44](#_Toc1571786)

[3.13.4 aggregateIds property 45](#_Toc1571787)

[3.13.5 baselineInstanceGuid property 45](#_Toc1571788)

[3.13.6 tool property 45](#_Toc1571789)

[3.13.7 invocations property 45](#_Toc1571790)

[3.13.8 conversion property 45](#_Toc1571791)

[3.13.9 versionControlProvenance property 45](#_Toc1571792)

[3.13.10 originalUriBaseIds property 46](#_Toc1571793)

[3.13.11 files property 47](#_Toc1571794)

[3.13.12 logicalLocations property 48](#_Toc1571795)

[3.13.13 graphs property 49](#_Toc1571796)

[3.13.14 results property 49](#_Toc1571797)

[3.13.15 resources property 49](#_Toc1571798)

[3.13.16 defaultFileEncoding property 50](#_Toc1571799)

[3.13.17 defaultSourceLanguage property 50](#_Toc1571800)

[3.13.18 newlineSequences property 50](#_Toc1571801)

[3.13.19 columnKind property 51](#_Toc1571802)

[3.13.20 richMessageMimeType property 51](#_Toc1571803)

[3.13.21 redactionToken property 51](#_Toc1571804)

[3.14 externaPropertyFile object 52](#_Toc1571805)

[3.14.1 General 52](#_Toc1571806)

[3.14.2 fileLocation property 52](#_Toc1571807)

[3.14.3 instanceGuid property 52](#_Toc1571808)

[3.14.4 itemCount property 52](#_Toc1571809)

[3.15 runAutomationDetails object 52](#_Toc1571810)

[3.15.1 General 52](#_Toc1571811)

[3.15.2 Constraints 53](#_Toc1571812)

[3.15.3 description property 53](#_Toc1571813)

[3.15.4 instanceId property 53](#_Toc1571814)

[3.15.5 instanceGuid property 54](#_Toc1571815)

[3.15.6 correlationGuid property 54](#_Toc1571816)

[3.16 tool object 54](#_Toc1571817)

[3.16.1 General 54](#_Toc1571818)

[3.16.2 name property 55](#_Toc1571819)

[3.16.3 fullName property 55](#_Toc1571820)

[3.16.4 semanticVersion property 55](#_Toc1571821)

[3.16.5 version property 55](#_Toc1571822)

[3.16.6 dottedQuadFileVersion property 55](#_Toc1571823)

[3.16.7 downloadUri property 56](#_Toc1571824)

[3.16.8 language property 56](#_Toc1571825)

[3.16.9 resourceLocation property 56](#_Toc1571826)

[3.16.10 sarifLoggerVersion property 57](#_Toc1571827)

[3.17 invocation object 57](#_Toc1571828)

[3.17.1 General 57](#_Toc1571829)

[3.17.2 commandLine property 57](#_Toc1571830)

[3.17.3 arguments property 58](#_Toc1571831)

[3.17.4 responseFiles property 58](#_Toc1571832)

[3.17.5 attachments property 59](#_Toc1571833)

[3.17.6 startTimeUtc property 60](#_Toc1571834)

[3.17.7 endTimeUtc property 60](#_Toc1571835)

[3.17.8 exitCode property 60](#_Toc1571836)

[3.17.9 exitCodeDescription property 60](#_Toc1571837)

[3.17.10 exitSignalName property 61](#_Toc1571838)

[3.17.11 exitSignalNumber property 61](#_Toc1571839)

[3.17.12 processStartFailureMessage property 61](#_Toc1571840)

[3.17.13 toolExecutionSuccessful property 61](#_Toc1571841)

[3.17.14 machine property 62](#_Toc1571842)

[3.17.15 account property 62](#_Toc1571843)

[3.17.16 processId property 62](#_Toc1571844)

[3.17.17 executableLocation property 62](#_Toc1571845)

[3.17.18 workingDirectory property 62](#_Toc1571846)

[3.17.19 environmentVariables property 62](#_Toc1571847)

[3.17.20 toolNotifications property 63](#_Toc1571848)

[3.17.21 configurationNotifications property 63](#_Toc1571849)

[3.17.22 stdin, stdout, stderr, and stdoutStderr properties 64](#_Toc1571850)

[3.18 attachment object 64](#_Toc1571851)

[3.18.1 General 64](#_Toc1571852)

[3.18.2 description property 65](#_Toc1571853)

[3.18.3 fileLocation property 65](#_Toc1571854)

[3.18.4 regions property 65](#_Toc1571855)

[3.18.5 rectangles property 65](#_Toc1571856)

[3.19 conversion object 66](#_Toc1571857)

[3.19.1 General 66](#_Toc1571858)

[3.19.2 tool property 66](#_Toc1571859)

[3.19.3 invocation property 66](#_Toc1571860)

[3.19.4 analysisToolLogFiles property 66](#_Toc1571861)

[3.20 versionControlDetails object 67](#_Toc1571862)

[3.20.1 General 67](#_Toc1571863)

[3.20.2 Constraints 67](#_Toc1571864)

[3.20.3 repositoryUri property 67](#_Toc1571865)

[3.20.4 revisionId property 67](#_Toc1571866)

[3.20.5 branch property 67](#_Toc1571867)

[3.20.6 revisionTag property 67](#_Toc1571868)

[3.20.7 asOfTimeUtc property 68](#_Toc1571869)

[3.20.8 mappedTo property 68](#_Toc1571870)

[3.21 file object 70](#_Toc1571871)

[3.21.1 General 70](#_Toc1571872)

[3.21.2 fileLocation property 70](#_Toc1571873)

[3.21.3 parentIndex property 70](#_Toc1571874)

[3.21.4 offset property 71](#_Toc1571875)

[3.21.5 length property 71](#_Toc1571876)

[3.21.6 roles property 71](#_Toc1571877)

[3.21.7 mimeType property 72](#_Toc1571878)

[3.21.8 contents property 72](#_Toc1571879)

[3.21.9 encoding property 72](#_Toc1571880)

[3.21.10 sourceLanguage property 73](#_Toc1571881)

[3.21.10.1 General 73](#_Toc1571882)

[3.21.10.2 Source language identifier conventions and practices 73](#_Toc1571883)

[3.21.11 hashes property 74](#_Toc1571884)

[3.21.12 lastModifiedTimeUtc property 75](#_Toc1571885)

[3.22 result object 75](#_Toc1571886)

[3.22.1 General 75](#_Toc1571887)

[3.22.2 Distinguishing logically identical from logically distinct results 75](#_Toc1571888)

[3.22.3 instanceGuid property 76](#_Toc1571889)

[3.22.4 correlationGuid property 76](#_Toc1571890)

[3.22.5 ruleId property 76](#_Toc1571891)

[3.22.6 ruleIndex property 77](#_Toc1571892)

[3.22.7 kind property 77](#_Toc1571893)

[3.22.8 level property 78](#_Toc1571894)

[3.22.9 message property 80](#_Toc1571895)

[3.22.10 locations property 81](#_Toc1571896)

[3.22.11 analysisTarget property 82](#_Toc1571897)

[3.22.12 fingerprints property 82](#_Toc1571898)

[3.22.13 partialFingerprints property 83](#_Toc1571899)

[3.22.14 codeFlows property 84](#_Toc1571900)

[3.22.15 graphs property 85](#_Toc1571901)

[3.22.16 graphTraversals property 85](#_Toc1571902)

[3.22.17 stacks property 85](#_Toc1571903)

[3.22.18 relatedLocations property 85](#_Toc1571904)

[3.22.19 suppressionStates property 86](#_Toc1571905)

[3.22.19.1 General 86](#_Toc1571906)

[3.22.19.2 suppressedInSource value 87](#_Toc1571907)

[3.22.19.3 suppressedExternally value 87](#_Toc1571908)

[3.22.20 baselineState property 87](#_Toc1571909)

[3.22.21 rank property 88](#_Toc1571910)

[3.22.22 attachments property 88](#_Toc1571911)

[3.22.23 workItemUris property 88](#_Toc1571912)

[3.22.24 hostedViewerUri property 89](#_Toc1571913)

[3.22.25 provenance property 89](#_Toc1571914)

[3.22.26 fixes property 89](#_Toc1571915)

[3.22.27 occurrenceCount property 89](#_Toc1571916)

[3.23 location object 89](#_Toc1571917)

[3.23.1 General 89](#_Toc1571918)

[3.23.2 Constraints 90](#_Toc1571919)

[3.23.3 physicalLocation property 90](#_Toc1571920)

[3.23.4 fullyQualifiedLogicalName property 90](#_Toc1571921)

[3.23.5 logicalLocationIndex property 92](#_Toc1571922)

[3.23.6 message property 92](#_Toc1571923)

[3.23.7 annotations property 92](#_Toc1571924)

[3.24 physicalLocation object 93](#_Toc1571925)

[3.24.1 General 93](#_Toc1571926)

[3.24.2 id property 93](#_Toc1571927)

[3.24.3 fileLocation property 93](#_Toc1571928)

[3.24.4 region property 93](#_Toc1571929)

[3.24.5 contextRegion property 94](#_Toc1571930)

[3.25 region object 94](#_Toc1571931)

[3.25.1 General 94](#_Toc1571932)

[3.25.2 Text regions 95](#_Toc1571933)

[3.25.3 Binary regions 97](#_Toc1571934)

[3.25.4 Independence of text and binary regions 97](#_Toc1571935)

[3.25.5 startLine property 98](#_Toc1571936)

[3.25.6 startColumn property 98](#_Toc1571937)

[3.25.7 endLine property 98](#_Toc1571938)

[3.25.8 endColumn property 98](#_Toc1571939)

[3.25.9 charOffset property 98](#_Toc1571940)

[3.25.10 charLength property 98](#_Toc1571941)

[3.25.11 byteOffset property 99](#_Toc1571942)

[3.25.12 byteLength property 99](#_Toc1571943)

[3.25.13 snippet property 99](#_Toc1571944)

[3.25.14 message property 99](#_Toc1571945)

[3.25.15 sourceLanguage property 99](#_Toc1571946)

[3.26 rectangle object 100](#_Toc1571947)

[3.26.1 General 100](#_Toc1571948)

[3.26.2 top, left, bottom, and right properties 100](#_Toc1571949)

[3.26.3 message property 100](#_Toc1571950)

[3.27 logicalLocation object 100](#_Toc1571951)

[3.27.1 General 100](#_Toc1571952)

[3.27.2 Logical location naming rules 100](#_Toc1571953)

[3.27.3 name property 101](#_Toc1571954)

[3.27.4 fullyQualifiedName property 101](#_Toc1571955)

[3.27.5 decoratedName property 102](#_Toc1571956)

[3.27.6 kind property 102](#_Toc1571957)

[3.27.7 parentIndex property 102](#_Toc1571958)

[3.28 codeFlow object 103](#_Toc1571959)

[3.28.1 General 103](#_Toc1571960)

[3.28.2 message property 104](#_Toc1571961)

[3.28.3 threadFlows property 104](#_Toc1571962)

[3.29 threadFlow object 104](#_Toc1571963)

[3.29.1 General 104](#_Toc1571964)

[3.29.2 id property 104](#_Toc1571965)

[3.29.3 message property 104](#_Toc1571966)

[3.29.4 locations property 104](#_Toc1571967)

[3.30 graph object 105](#_Toc1571968)

[3.30.1 General 105](#_Toc1571969)

[3.30.2 id property 105](#_Toc1571970)

[3.30.3 description property 105](#_Toc1571971)

[3.30.4 nodes property 105](#_Toc1571972)

[3.30.5 edges property 105](#_Toc1571973)

[3.31 node object 106](#_Toc1571974)

[3.31.1 General 106](#_Toc1571975)

[3.31.2 id property 106](#_Toc1571976)

[3.31.3 label property 106](#_Toc1571977)

[3.31.4 location property 106](#_Toc1571978)

[3.31.5 children property 106](#_Toc1571979)

[3.32 edge object 107](#_Toc1571980)

[3.32.1 General 107](#_Toc1571981)

[3.32.2 id property 107](#_Toc1571982)

[3.32.3 label property 107](#_Toc1571983)

[3.32.4 sourceNodeId property 107](#_Toc1571984)

[3.32.5 targetNodeId property 107](#_Toc1571985)

[3.33 graphTraversal object 108](#_Toc1571986)

[3.33.1 General 108](#_Toc1571987)

[3.33.2 graphId property 108](#_Toc1571988)

[3.33.3 description property 108](#_Toc1571989)

[3.33.4 initialState property 108](#_Toc1571990)

[3.33.5 edgeTraversals property 108](#_Toc1571991)

[3.34 edgeTraversal object 110](#_Toc1571992)

[3.34.1 General 110](#_Toc1571993)

[3.34.2 edgeId property 110](#_Toc1571994)

[3.34.3 message property 110](#_Toc1571995)

[3.34.4 finalState property 110](#_Toc1571996)

[3.34.5 stepOverEdgeCount property 110](#_Toc1571997)

[3.35 stack object 111](#_Toc1571998)

[3.35.1 General 111](#_Toc1571999)

[3.35.2 message property 112](#_Toc1572000)

[3.35.3 frames property 112](#_Toc1572001)

[3.36 stackFrame object 112](#_Toc1572002)

[3.36.1 General 112](#_Toc1572003)

[3.36.2 location property 112](#_Toc1572004)

[3.36.3 module property 112](#_Toc1572005)

[3.36.4 threadId property 112](#_Toc1572006)

[3.36.5 address property 112](#_Toc1572007)

[3.36.6 offset property 112](#_Toc1572008)

[3.36.7 parameters property 113](#_Toc1572009)

[3.37 threadFlowLocation object 113](#_Toc1572010)

[3.37.1 General 113](#_Toc1572011)

[3.37.2 location property 113](#_Toc1572012)

[3.37.3 module property 114](#_Toc1572013)

[3.37.4 stack property 114](#_Toc1572014)

[3.37.5 kind property 114](#_Toc1572015)

[3.37.6 state property 114](#_Toc1572016)

[3.37.7 nestingLevel property 115](#_Toc1572017)

[3.37.8 executionOrder property 115](#_Toc1572018)

[3.37.9 executionTimeUtc property 116](#_Toc1572019)

[3.37.10 importance property 116](#_Toc1572020)

[3.38 resultProvenance object 116](#_Toc1572021)

[3.38.1 General 116](#_Toc1572022)

[3.38.2 firstDetectionTimeUtc property 116](#_Toc1572023)

[3.38.3 lastDetectionTimeUtc property 117](#_Toc1572024)

[3.38.4 firstDetectionRunInstanceGuid property 117](#_Toc1572025)

[3.38.5 lastDetectionRunInstanceGuid property 117](#_Toc1572026)

[3.38.6 invocationIndex property 117](#_Toc1572027)

[3.38.7 conversionSources property 118](#_Toc1572028)

[3.39 resources object 119](#_Toc1572029)

[3.39.1 General 119](#_Toc1572030)

[3.39.2 messageStrings property 119](#_Toc1572031)

[3.39.3 rules property 119](#_Toc1572032)

[3.40 rule object 120](#_Toc1572033)

[3.40.1 General 120](#_Toc1572034)

[3.40.2 Constraints 120](#_Toc1572035)

[3.40.3 id property 120](#_Toc1572036)

[3.40.4 deprecatedIds property 120](#_Toc1572037)

[3.40.5 name property 122](#_Toc1572038)

[3.40.6 shortDescription property 123](#_Toc1572039)

[3.40.7 fullDescription property 123](#_Toc1572040)

[3.40.8 messageStrings property 123](#_Toc1572041)

[3.40.9 richMessageStrings property 124](#_Toc1572042)

[3.40.10 helpUri property 124](#_Toc1572043)

[3.40.11 help property 124](#_Toc1572044)

[3.40.12 configuration property 124](#_Toc1572045)

[3.41 ruleConfiguration object 124](#_Toc1572046)

[3.41.1 General 124](#_Toc1572047)

[3.41.2 enabled property 125](#_Toc1572048)

[3.41.3 defaultLevel property 125](#_Toc1572049)

[3.41.4 defaultRank property 125](#_Toc1572050)

[3.41.5 parameters property 125](#_Toc1572051)

[3.42 fix object 126](#_Toc1572052)

[3.42.1 General 126](#_Toc1572053)

[3.42.2 description property 126](#_Toc1572054)

[3.42.3 fileChanges property 126](#_Toc1572055)

[3.43 fileChange object 128](#_Toc1572056)

[3.43.1 General 128](#_Toc1572057)

[3.43.2 fileLocation property 128](#_Toc1572058)

[3.43.3 replacements property 128](#_Toc1572059)

[3.44 replacement object 129](#_Toc1572060)

[3.44.1 General 129](#_Toc1572061)

[3.44.2 Constraints 130](#_Toc1572062)

[3.44.3 deletedRegion property 130](#_Toc1572063)

[3.44.4 insertedContent property 130](#_Toc1572064)

[3.45 notification object 131](#_Toc1572065)

[3.45.1 General 131](#_Toc1572066)

[3.45.2 id property 131](#_Toc1572067)

[3.45.3 ruleId property 131](#_Toc1572068)

[3.45.4 ruleIndex property 131](#_Toc1572069)

[3.45.5 physicalLocation property 132](#_Toc1572070)

[3.45.6 message property 132](#_Toc1572071)

[3.45.7 level property 132](#_Toc1572072)

[3.45.8 threadId property 132](#_Toc1572073)

[3.45.9 timeUtc property 132](#_Toc1572074)

[3.45.10 exception property 133](#_Toc1572075)

[3.46 exception object 133](#_Toc1572076)

[3.46.1 General 133](#_Toc1572077)

[3.46.2 kind property 133](#_Toc1572078)

[3.46.3 message property 133](#_Toc1572079)

[3.46.4 stack property 133](#_Toc1572080)

[3.46.5 innerExceptions property 133](#_Toc1572081)

[4 External property file format 135](#_Toc1572082)

[4.1 General 135](#_Toc1572083)

[4.2 External property file naming convention 135](#_Toc1572084)

[4.3 externalProperties object 135](#_Toc1572085)

[4.3.1 General 135](#_Toc1572086)

[4.3.2 $schema property 136](#_Toc1572087)

[4.3.3 version property 136](#_Toc1572088)

[4.3.4 instanceGuid property 136](#_Toc1572089)

[4.3.5 runInstanceGuid property 136](#_Toc1572090)

[4.3.6 The property value properties 136](#_Toc1572091)

[5 Conformance 137](#_Toc1572092)

[5.1 Conformance targets 137](#_Toc1572093)

[5.2 Conformance Clause 1: SARIF log file 137](#_Toc1572094)

[5.3 Conformance Clause 2: SARIF resource file 137](#_Toc1572095)

[5.4 Conformance Clause 3: SARIF producer 137](#_Toc1572096)

[5.5 Conformance Clause 4: Direct producer 138](#_Toc1572097)

[5.6 Conformance Clause 5: Deterministic producer 138](#_Toc1572098)

[5.7 Conformance Clause 6: Converter 138](#_Toc1572099)

[5.8 Conformance Clause 7: SARIF post-processor 138](#_Toc1572100)

[5.9 Conformance Clause 8: SARIF consumer 138](#_Toc1572101)

[5.10 Conformance Clause 9: Viewer 138](#_Toc1572102)

[5.11 Conformance Clause 10: Result management system 138](#_Toc1572103)

[5.12 Conformance Clause 11: Engineering system 139](#_Toc1572104)

[Appendix A. (Informative) Acknowledgments 140](#_Toc1572105)

[Appendix B. (Normative) Use of fingerprints by result management systems 141](#_Toc1572106)

[Appendix C. (Informative) Use of SARIF by log file viewers 142](#_Toc1572107)

[Appendix D. (Informative) Production of SARIF by converters 143](#_Toc1572108)

[Appendix E. (Informative) Locating rule metadata 144](#_Toc1572109)

[Appendix F. (Normative) Producing deterministic SARIF log files 145](#_Toc1572110)

[F.1 General 145](#_Toc1572111)

[F.2 Non-deterministic file format elements 145](#_Toc1572112)

[F.3 Array and dictionary element ordering 146](#_Toc1572113)

[F.4 Absolute paths 146](#_Toc1572114)

[F.5 Compensating for non-deterministic output 146](#_Toc1572115)

[F.6 Interaction between determinism and baselining 147](#_Toc1572116)

[Appendix G. (Informative) Guidance on fixes 148](#_Toc1572117)

[Appendix H. (Informative) Diagnosing results in generated files 149](#_Toc1572118)

[Appendix I. (Informative) Sample sourceLanguage values 152](#_Toc1572119)

[Appendix J. (Informative) Examples 154](#_Toc1572120)

[J.1 Minimal valid SARIF log file 154](#_Toc1572121)

[J.2 Minimal recommended SARIF log file with source information 154](#_Toc1572122)

[J.3 Minimal recommended SARIF log file without source information 155](#_Toc1572123)

[J.4 SARIF resource file with rule metadata 156](#_Toc1572124)

[J.5 Comprehensive SARIF file 157](#_Toc1572125)

[Appendix K. (Informative) Revision History 166](#_Toc1572126)

# 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, called the Static Analysis Results Interchange Format, or “SARIF”[[1]](#footnote-1). 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 location in 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 [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](#def_run) to a baseline produced by a [baseline run](#def_baseline_run) to determine whether new results have been introduced.

baseline run

[run](#def_run) that produces a [baseline](#def_baseline) to which subsequent runs can be compared

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

set of one or more [thread flows](#def_thread_flow) which together specify a pattern of code execution relevant to detecting a [result](#def_result)

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

deterministic producer

[SARIF producer](#def_sarif_producer) which, given identical inputs, repeatedly produces an identical [SARIF log file](#def_sarif_log_file)

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)

engineering system

software development environment within which [analysis tools](#def_static_analysis_tool) execute

NOTE: An engineering system might include a build system, a source control system, a [result management system](#def_result_management_system), a bug tracking system, a test execution system, and so on.

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 property

property stored outside of the [SARIF log file](#def_log_file) to which it logically belongs

external property file

file containing the values of one or more [external properties](#def_external_property)

externalizable property

property that can be contained in an [external property file](#def_external_property_file)

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 addressable *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

fully qualified logical name

string that that fully identifies the programmatic construct specified by a [logical location](#def_logical_location), typically by means of a hierarchical identifier.

Example: The fully qualified logical name of the C# method f(void) in class C in namespace N is "N.C.f(void)". Its [logical name](#def_logical_name) is "f(void)".

hierarchical string

string in the format <component>{/<component>}\*, for example, "CWE/22"

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.

logical name

string that partially identifies the programmatic construct specified by a [logical location](#def_logical_location) by specifying most specific (often the rightmost) component of its [fully qualified logical name](#def_fully_qualified_logical_name).

Example: The logical name of the C# method f(void) in class C in namespace N is "f(void)". Its [fully qualified logical name](#def_fully_qualified_logical_name) is "N.C.f(void)".

message string

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

nested file

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

nested logical location

[logical location](#def_logical_location) that is contained 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

attribute of an object consisting of a name and a value associated with the name

property bag

object consisting of an unordered set of non-standardized [properties](#def_property) with arbitrary [camelCase names](#def_camelCase_name)

redactable property

[property](#def_property) that potentially contains sensitive information that a SARIF [direct producer](#def_direct_producer) or a [SARIF post-processor](#def_post_processor) might wish to redact

region

contiguous portion of a [file](#def_file)

repository

container for a related set of files in a version control system

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 [analysis tools](#def_static_analysis_tool), produces reports that enable engineering 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.

rich text message

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

root file

[SARIF log file](#def_log_file) to which one or more [external property files](#def_external_property_file) logically belong

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](#def_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 post-processor

[SARIF producer](#def_sarif_producer) that transforms an existing [SARIF log file](#def_sarif_log_file) into a new SARIF log file, for example, by removing or redacting security-sensitive elements.

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)

thread flow

temporally ordered set of code locations specifying a possible execution path through the code, which occur within a single thread of execution, such as an operating system thread or a fiber

top-level file

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

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).

VCS

version control system

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/>.

[IANA-ENC] Freed, Ned and Dürst, Martin, “Character Sets”, 2017-12-20, <https://www.iana.org/assignments/character-sets/character-sets.xhtml>.

[IANA-HASH] “Hash Function Textual Names”, <https://www.iana.org/assignments/hash-function-text-names/hash-function-text-names.xhtml>, July 4, 2017.

[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>.

[RFC3987] Duerst, M. and Suignard, M., "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/info/rfc3987>.

[RFC4122] Leach, P., Mealling, M., and Salz, R., "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <http://www.rfc-editor.org/info/rfc4122>.

[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>.

[RFC8089] Kerwin, M., "The "file" URI Scheme", RFC 8089, DOI 10.17487/RFC8089, February 2017, <http://www.rfc-editor.org/info/rfc8089>.

[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>.

[ISO14882:2017] “Information technology – Programming languages – C++”, ISO/IEC 14882, December 2017, <https://www.iso.org/standard/68564.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 JSON serialization 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 SARIF 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 charLength property. For clarity, we can refer to the charLength property as physicalLocation.region.charLength.

## 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

SARIF defines an object model, the top level of which is the sarifLog object (§3.12), which contains 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** contain a serialization of the SARIF object model into the JSON format.

NOTE 1: In the future, other serializations might be defined.

The top-level value in the log file, representing the sarifLog object, **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)].

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

NOTE 2: [[RFC8259](#RFC8259)] requires this encoding for any JSON text “exchanged between systems that are not part of a closed ecosystem.”

## SARIF file naming convention

The file name of a SARIF log file **SHOULD** end with the extension ".sarif".

EXAMPLE 1: output.sarif

The file name **MAY** end with the additional extension ".json".

EXAMPLE 2: output.sarif.json

## 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.

Notwithstanding any necessary transcoding and escaping, the SARIF producer **SHALL** preserve the text file’s line breaking convention (for example, "\n" or "\r\n").

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 (§3.4.3). If the uri property contains a relative reference (the term used in [[RFC 3986](#RFC3986)] for what is commonly called a “relative URI”), the uriBaseId property (§3.4.4) can sometimes be used to resolve the relative reference to an absolute URI.

### Constraints

At least one of the uri property (§3.4.3) or the fileIndex property (§3.4.5) **SHALL** be present. In certain circumstances (see §3.4.4 and §3.4.5), they **MAY** both be present.

NOTE: Providing both uri and fileIndex makes the log file more readable at the expense of increased size. Providing only fileIndex reduces log file size but makes it less readable to an end user, who has to determine the URI by locating the file object (§3.21) at the index within run.files (§3.13, §3.13.11) specified by fileIndex.

If both uri and fileIndex are present, they **SHALL** both denote the same file. That is, let URI1 be the fully resolved URI of the file specified by a fileLocation object as determined by the uriBaseId resolution procedure described in §3.4.4. Let URI2 be the fully resolved URI of the file specified by the file object (§3.21) indicated by fileIndex, determined in the same way. Then URI1 and URI2 **SHALL** be equivalent in the sense described in §3.10.1.

### uri property

Depending on the circumstances, a fileLocation object either **SHALL**, **SHALL NOT**, or **MAY** contain a property named uri whose value is a string containing a URI [[RFC3986](#RFC3986)] that specifies the location of the file. If the fileLocation object describes a nested file whose location within its parent container can be expressed by a path from the root of the container, then if uri is present, it **SHALL** specify a relative reference [[RFC3986](#RFC3986)] expressing that path.

If the fileIndex property (§3.4.5) is absent, uri **SHALL** be present.

NOTE 1: This ensures that there is a way to locate the file specified by the fileLocation object.

If the fileLocation object represents a nested file whose location within its parent container can be expressed only by means of a byte offset, then uri **SHALL NOT** be present.

NOTE 2: This implies that fileIndex will be present; see §3.4.5.

Otherwise, uri **MAY** be present.

### uriBaseId property

If this fileLocation object describes a top-level file and the value of its uri property (§3.4.3) is a relative reference, the fileLocation object **SHOULD** contain a property named uriBaseId whose value is a string which indirectly specifies the absolute URI with respect to which that relative reference is interpreted. If the uri property contains an absolute URI, the uriBaseId property **SHALL** be absent.If this fileLocation object describes a nested file, uriBaseId **SHALL** be absent.

If a SARIF consumer requires an absolute URI (for example, to display the specified file to a user), then it needs to resolve uriBaseId to an absolute URI, which it can then combine with the relative reference stored in the uri property.

A SARIF consumer **SHALL** use the following procedure to resolve a uriBaseId to an absolute URI:

1. If the end user has configured the SARIF consumer with a value for the uriBaseId (for example, on the consumer’s command line or through a user interface prompt), then the consumer **SHALL** use the configured value.

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/"

1. If uriBaseId is not yet resolved and the originalUriBaseIds property (§3.13.10) of the containing run object (§3.13) is present, the consumer **SHALL** attempt to resolve the uriBaseId from the information in originalUriBaseIds, in the manner specified in §3.13.10.
2. If uriBaseId is not yet resolved, the consumer **MAY** use other information or heuristics to locate the file.

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 a fileLocation object (§3.4) to a relative reference. The tool has also set the uriBaseId property to "%srcroot%". The analysis tool and the SARIF consumers have agreed upon a convention whereby this indicates that the relative reference is expressed relative to the root of the source tree in which the file appears.

"fileLocation": {

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

"uriBaseId": "%srcroot%"

}

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

* Portability: A log file that contains relative references 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](#AppendixDeterminism), “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”.

For more guidance on the intended use of the uriBaseId property, see §3.4.6.

### fileIndex property

Depending on the circumstances, a fileLocation object either **SHALL**, **SHALL NOT**, or **SHOULD** contain a property named fileIndex whose value is a non-negative integer that specifies the index within the array-valued files property (§3.13.11) of the containing run object (§3.13) of the file object (§3.21), if any, that describes the file specified by this fileLocation object. If fileIndex is absent, it **SHALL** default to -1 (which is otherwise not a valid value for this property).

If the uri property (§3.4.3) is absent, then fileIndex **SHALL** be present.

NOTE 1: This ensures that there is a way to locate the file specified by the fileLocation object.

If run.files is absent or does not contain an element that describes the file specified by this fileLocation object, then fileIndex **SHALL NOT** be present.

NOTE 2: This implies that uri is present.

Otherwise, fileIndex **SHOULD** be present.

NOTE 3: If fileIndex is absent, the SARIF consumer will not be able to locate the information contained in the file object about the file specified by this fileLocation object.

EXAMPLE: In this example, results[0].locations[0].physicalLocation.fileLocation.fileIndex specifies the file object located at files[0].

{ # A run object (§3.13).

"files": [

{

"fileLocation": {

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

},

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

}

],

"results": [

{

"ruleId": "CA2101",

"level": "error",

"locations": [

{

"physicalLocation": {

"fileLocation": {

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

"fileIndex": 0

},

"region: {

"startLine": 24,

"startColumn": 9

}

}

}

]

}

]

}

### Guidance on the use of fileLocation objects

Some URIs are “deterministic” in the sense that they will be the same from one run to the next and are independent of machine-specific information such as volume names or drive letters. Internet addresses are typically deterministic.

In contrast, file system paths are typically non-deterministic. For example, a source code enlistment might exist at different paths on different machines.

fileLocation objects represent non-deterministic URIs. **The** uri property (§3.4.3) **SHOULD** contain a relative reference that *is* deterministic, for example, the relative path from the root of a source code enlistment to the file. The uriBaseId property (§3.4.4) **SHOULD** capture the non-deterministic portion of the URI, for example, the absolute path to the root of the source code enlistment.

EXAMPLE: In this example, the location of a result detected by a tool is specified by a relative reference together with a uriBaseId that specifies the root of the source code enlistment.

{ # A run object (§3.13).

"originalUriBaseIds": { # See §3.13.10.

"SRCROOT": {

"uri": "file:///C:/browser/src/"

}

},

"results": [ # See §3.13.14.

{ # A result object (§3.22).

"locations": [ # See §3.22.10.

{ # A location object (§3.23).

"physicalLocation": { # See §3.23.3.

"fileLocation": { # A fileLocation object.

"uri": "ui/window.cpp",

"uriBaseId": "SRCROOT"

}

}

}

]

}

]

}

## String properties

### General

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

### Redactable string properties

Certain string-valued properties in this specification (for example, invocation.commandLine (§3.17.2)) might contain sensitive information that a SARIF producer or a SARIF post-processor might choose to redact. We describe these properties as being “redactable.” The description of every redactable property will state that it is redactable.

If a SARIF producer or a SARIF post-processor chooses to redact sensitive information in a redactable property, it **SHALL** replace the sensitive information with the string whose value is provided by the run.redactionToken property (§3.13.21).

### GUID-valued string properties

Certain string-valued properties in this specification provide unique stable identifiers in the form of a GUID or UUID [[RFC4122](#RFC4122)]. This document uses the term “GUID”.

EXAMPLE: "f81d4fae-7dec-11d0-a765-00a0c91e6bf6"

NOTE: RFC4122 allows hex digits in either upper or lower case. It does not permit delimiters such as curly braces ("{", "}") around the value.

The description of every GUID-valued property will state that it is GUID-valued.

### Hierarchical strings

#### General

Certain string-valued properties and certain property names in this specification (for example, the value of the runAutomationDetails.instanceId property (§3.15.4), and the property names in a property bag (§3.8)) are said to be “hierarchical”. This means that the string consists of a sequence of forward-slash-separated components, with this syntax:

hierarchical string = component, { "/", component };

component = { component character };

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

NOTE 1: The grammar prohibits a component from containing a forward slash. There is no escape mechanism to allow a component to include a forward slash.

For examples, see §3.8.2 and §3.15.4.

The description of every hierarchical string will state that it is hierarchical.

A SARIF consumer **SHALL** interpret the values of a hierarchical string as forming a logical hierarchy. The first component represents the top level of the hierarchy, the second component represents the second level, and so on.

NOTE 2: A hierarchical string does not need to include any forward slashes. The syntax permits a single string of non-forward-slash characters. The purpose of this section is to define the semantics of the forward slash character in those properties that respect it.

In string-valued properties and property names that are *not* described as hierarchical, the forward slash character has no special meaning, and a SARIF consumer **SHALL NOT** interpret it as dividing the value into hierarchical components.

#### Versioned hierarchical strings

Certain hierarchical strings in this specification (for example, the property names in result.fingerprints (§3.22.12) and result.partialFingerprints (§3.22.13)) are said to be “versioned.” This means that if the last component of the string is of the form

version component = "v", non negative integer;

then a SARIF consumer **SHALL** consider that component to represent the version number of the entity specified by the string.

The description of every versioned hierarchical string will state that it is versioned.

In string-valued properties and property names that are described as hierarchical but *not* as versioned, a final component matching the syntax of version component has no special meaning, and a SARIF consumer **SHALL NOT** interpret it as a version number.

NOTE: A versioned hierarchical string does not need to include a version component. The syntax permits but does not require it.

A hierarchical string without a version component **SHALL** be considered older than any corresponding string with a version component.

EXAMPLE: In this example, the partial fingerprint whose property name is "prohibitedWordHash" is considered to have been computed with an older version of the “prohibited word hash” algorithm than the partial fingerprint whose property name is "prohibitedWordHash/v1".

{ # A result object (§3.22).

"partialFingerprints": { # See §3.22.13.

"prohibitedWordHash": "4efcc21977b55",

"prohibitedWordHash/v1": "097886bc876fe"

}

}

NOTE: When a previously unversioned string is later versioned, as in the example above, it might be clearer to specify "v2" for the first explicitly versioned string.

## Object properties

Certain properties in this specification are defined to be objects whose property names satisfy certain conditions. Examples are run.files (§3.13.11) and rule.messageStrings (§3.40.8). 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 arrays. Examples are the invocation.toolNotifications property (§3.17.20) and the property bag tags property (§3.8.2).

### 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”. In particular, two strings are considered equal when they consist of the same sequence of Unicode [[UNICODE10](#UNICODE10)] code points.

## Property bags

### General

Certain properties in this specification are defined to be “property bags”. A property bag is an object (§3.6) containing an unordered set of properties with arbitrary names.

The property names are hierarchical strings (§3.5.4). The components of the property names **SHOULD** be camelCase strings, but see [Appendix D](#AppendixConverters) for exceptions.

The property values **MAY** be of any JSON type, including strings, numbers, arrays, objects, Booleans, and null. If a property value is a string, it **MAY** be an empty string.

In addition to those properties that are explicitly documented, every object defined in this specification **MAY** contain a property named properties whose value is a property bag. This allows SARIF producers to include information about each object that is not explicitly specified in the SARIF format.

### Tags

#### General

If a property bag contains a property named tags, the property value **SHALL** be an array of zero or more unique strings (§3.7.2). Two strings **SHALL** be considered the same if they consist of the same sequence of Unicode [[UNICODE10](#UNICODE10)] code points.

If tags is absent, it **SHALL** default to an empty array.

The strings in the tags array are hierarchical (§3.5.4).

EXAMPLE: In this example, the SARIF producer categorizes scan results according to the Common Weakness Enumeration taxonomy [[CWE](#CWE)].

{ # A result object (§3.22).

"ruleId": "CA2124",

...

"properties": {

"tags": [

"CWE/22"

]

}

}

#### 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 (§3.22)

"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 (in this example, the property bag belonging to the result object):

{ # A result object (§3.22)

"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 (§3.13) that lexically contains the result object:

{ # A run object (see §3.13)

"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

The time component of every date/time-valued property **SHALL** be expressed in Coordinated Universal Time (UTC).

NOTE 1: The name of every date/time-valued property ends in “Utc” to emphasize thos requirement.

The time components of date/time-valued properties in property bags (§3.8) **SHOULD** also be expressed in UTC.

NOTE 2: This might not always be possible if the property comes from a source that does not provide time zone information.

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

## URI-valued properties

### General

Certain properties in this specification specify either an absolute URI or a URI reference (the term used in [[RFC3986](#RFC3986)] to describe either an absolute URI or a relative reference). The value of every such property, if present, **SHALL** be a string in the format specified by [[RFC3986](#RFC3986)].

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

NOTE 1: A URI reference (even a relative reference) 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.

The URI **SHALL** specify the location of the file at the time the analysis was performed.

Two URI references **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 [[RFC3986](#RFC3986)]:

* 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 URI references 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 a URI reference; for example, they **SHALL NOT** convert the 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 Microsoft 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 URI references in the log file would not match the path names on the destination system.

### URIs that use the file scheme

If a URI uses the "file" protocol [[RFC8089](#RFC8089)] and the specified path is network-accessible, the SARIF producer **SHALL** include the host name.

EXAMPLE 1: A file-based URI that references a network share.

file://build.example.com/drops/Build-2018-04-19.01/src

If a URI uses the "file" protocol and the specified path is *not* network-accessible, the SARIF producer **SHOULD NOT** include the host name.

EXAMPLE 2: A file-based URI that references the local file system.

file:///C:/src

A SARIF producer **MAY** choose to omit the hostname (authority) from a file URI, for example, for security reasons. If it does so, then to maximize interoperability with previous version of the URI specification, the URI **SHOULD** start with "file:///", as in EXAMPLE 2. See [[RFC8089](#RFC8089)] for more information on this point.

### Internationalized Resource Identifiers (IRIs)

If a URI-valued property refers to resource identified by an Internationalized Resource Identifier (IRI) [[RFC3987](#RFC3987)], the SARIF producer **SHALL** first transform the IRI into a URI, using the mapping mechanism specified in [[RFC3987](#RFC3987)] §3.1, and then assign the transformed value to the property. The string value of a URI-valued property **SHALL NOT** include Unicode characters such as "é"; such characters are permitted in IRIs but are not permitted in URIs. [[RFC3987](#RFC3987)] §3.1 describes how to replace such characters with “percent-encoded” equivalents to produce a valid URI.

EXAMPLE: Suppose a URI-valued property needs to refer to a resource identified by the string "http://www.example.com/hu/sör.txt". This string contains the character "ö", so it is a valid IRI but not a valid URI. Following the procedure in [[RFC3987](#RFC3987)] §3.1, a SARIF producer would transform this string to the valid URI "http://www.example.com/hu/s%C3%B6r.txt" before assigning it to the property.

## 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 (for example, "\r" or "\n"). However, if line breaks are present, they **MAY** follow any convention (for example, "\n" or "\r\n"). A SARIF post-processor **MAY** normalize line breaks to any desired convention, including escaping or removing the line breaks so that the entire message renders on a single line.

The message string **MAY** contain placeholders (§3.11.4) and embedded links (§3.11.5).

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 (§3.11.4) and embedded links (§3.11.5).

Every rich text message in a given run **SHALL** be expressed in the same markup language, specified by the run.richMessageMimeType property (§3.13.20). 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 (§3.11.11).

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 (§3.11.11). 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 1: Suppose a message object’s text property (§3.11.7) 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}.

EXAMPLE 2: In this example, the SARIF consumer will replace the placeholder {0} in message.text with the value "pBuffer" from the 0 element of message.arguments.

{ # A run object (§3.13).

"results": [ # See §3.13.14.

{ # A result object (§3.22).

"ruleId": "CA2101", # See §3.22.5.

"message": { # See §3.22.9.

"text": "Variable '{0}' is uninitialized.", # See §3.11.7.

"arguments": [ "pBuffer" ] # See §3.11.11.

}

}

]

}

### Messages with embedded links

A message string **MAY** include one or more links to locations within files mentioned in the enclosing result object (§3.22). We refer to these links as “embedded links”.

Within a rich text message (§3.11.3), an embedded link **SHALL** conform to the syntax of a GitHub Flavored Markdown link (see [[GFM](#GFM)], §6.6, “Links”), with the restriction that the “link destination” **SHALL** be a non-negative integer (whose interpretation is defined below).

NOTE: The GFM link syntax is very flexible. Since a SARIF viewer that renders rich text messages will presumably rely on a full-featured GFM processor, there is no need to restrict the embedded link syntax in SARIF rich text messages.

Within a plain text message (§3.11.2), an embedded link **SHALL** conform to the following syntax (which is a greatly restricted subset of the GFM link syntax) before JSON encoding:

escaped link character = "\" | "[" | "]";

normal link character = ? JSON string character ? – escaped link character;

link character = normal link character | ("\", escaped link character);

link text = { link character };

link destination = non negative integer;

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

link text is the message text visible to the user.

Literal square brackets ("[" and "]") in the link text of a plain text message **SHALL** be escaped with a backslash ("\").

NOTE: When a SARIF log file is serialized as JSON, JSON encoding doubles the backslash.

EXAMPLE 1: Consider this embedded link whose link text contains square brackets and backslashes:

"message": {

"text": "Prohibited term used in [para\\[0\\]\\\\spans\\[2\\](1)."

}

A SARIF viewer would render it as follows:

Prohibited term used in [para[0]\spans[2]](1).

Literal square brackets and (doubled) backslashes **MAY** appear anywhere else in a plain text message without being escaped.

The message object’s containing result object (§3.22) **SHALL** contain exactly one physicalLocation object (§3.24) whose id property (§3.24.2) equals the value of link destination.

NOTE: link destination 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 destination.

{

"version": "2.0.0",

"runs": [

{

"results": [

{

"ruleId": "TNT0001",

"message": {

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

},

"locations": [

{

"physicalLocation": {

"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 (§3.11.7) and richText (§3.11.8) properties. It can also indirectly refer to message strings through its messageId (§3.11.9) and richMessageId (§3.11.10) 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 (§3.13) can optionally contain the message string resources for a single language, namely the language designated by its tool.language property (§3.16.8). We refer to these message strings as “embedded resources.” Embedded message string resources are stored in the run.resources.messageStrings property (§3.39.2).

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.” §3.11.6.4 defines the naming convention and file lookup procedure for SARIF resources files. §3.11.6.5 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 naming convention

The file name of a SARIF resource file **SHALL** follow the naming convention defined by the following syntax:

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

language tag = ? RFC 5646 language tag ?;

EXAMPLE 1: en-US.sarif-resources

The file name **MAY** end with the additional extension ".json".

EXAMPLE 2: en-US.sarif-resources.json

#### 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.

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 (§3.16.9) 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 [[RFC5646](#RFC5646)] 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.sarif-resources or fr-FR.sarif-resources.json.) 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.sarif-resources or fr.sarif-resources.json.) 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 §3.11.6.2. 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 §3.11.6.2 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 (§3.12). 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 (§3.12.3) | string | No | Specifies the absolute URI from which the JSON schema for the SARIF resource file format (rather than the SARIF log file format) can be obtained. |
| runs (§3.12.4) | run[] (§3.13) | Yes | Array contains exactly one element, rather than one or more. That element contains only the properties specified in §3.11.6.5.3. |

##### 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 (§3.13.6) | tool (§3.16) | Yes | Required rather than optional. Contains only the properties specified in §3.11.6.5.4. |
| resources (§3.13.15) | resources (§3.39) | 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 (§3.16.2) | string | Yes | None |
| fullName (§3.16.3) | string | No | None |
| semanticVersion (§3.16.4) | string | Yes | None |
| version (§3.16.5) | string | No | None |
| dottedQuadFileVersion (§3.16.6) | string | No | None |
| language (§3.16.8) | 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 (§3.39).

### text property

A message object **MAY** contain a property named text whose value is a non-empty string containing a plain text message (§3.11.2).

### richText property

A message object **MAY** contain a property named richText whose value is a non-empty string containing a rich text message (§3.11.3).

If the richText property is present, the text property (§3.11.7) **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 (§3.11.6) for the desired plain text message (§3.11.2). See §3.11.6.2 and §3.11.6.4 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 (§3.11.6) for the desired rich text message (§3.11.3).

SARIF consumers that cannot (or choose not to) render rich text **SHALL** ignore the richMessageId property and use the messageId property instead. See §3.11.6.2 and §3.11.6.4 for details of the resource string lookup procedure.

### arguments property

If the message string specified by any of the properties text (§3.11.7), richText (§3.11.8), messageId (§3.11.9), or richMessageId (§3.11.10) contains any placeholders (§3.11.4), the message object **SHALL** contain a property named arguments whose value is an array of strings. §3.11.4 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, then arguments **MAY** be absent; if it is present, its value **SHALL** be an empty array.

If arguments is absent, it **SHALL** default to an empty array.

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 §3.12.2.

"runs": [ # See §3.12.4.

{

... # A run object (§3.13)

},

...

{

... # 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 specification 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.  
  
**NOTE: Throughout this specification, the version "2.0.0" is used. However, until this specification is finalized, producers SHOULD use the version value "2.0.0-csd.2.beta.YYYY-MM-DD" (found on the title page of this document) to denote the specific draft version to which their output conforms. This note will be removed in the final version of the specification.**

### $schema property

A sarifLog object **MAY** contain a property named $schema whose value is a string containing an absolute 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 (§3.12.2).

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 either null or an array of zero or more run objects (§3.13).

The value of runs **SHALL** be an array with at least one element except in the following circumstances:

* If the containing sarifLog object is the output of a query on a result management system, and the query did not match any runs stored in the result management system, then the value of runs **SHALL** be an empty array.
* If the containing sarifLog object is the output of a query on a result management system, and the result management system was unable to execute the query (for example, because the query was malformed) then the value of runs **SHALL** be null.

## run object

### General

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

EXAMPLE:

{

"tool": { # See §3.13.6.

... # A tool object (§3.16).

},

"results": [ # See §3.13.14.

{

... # A result object (§3.22).

},

...

{

... # Another result object.

}

]

}

### externalPropertyFiles property

#### Rationale

In some engineering environments, a single tool run might analyze hundreds of thousands of files and produce millions of results. This causes problems for both producers and consumers of such large SARIF log files:

* The log file might be too large for a consumer to hold in memory and might take several minutes to read.
* During production, some information (such as the complete set of files that were analyzed, the complete set of rules that were violated, or the end time of the run) cannot be known until the run is complete. Therefore, it is likely to be serialized at the end of the log file. However, consumers might need to access some of that information before reading the entire file. For example, a SARIF viewer might need to display rule metadata along with each result it displays, or to display the start and end times of a set of tool runs.

To mitigate these problems, SARIF allows certain properties of a run object to be stored in separate files. We refer to these files as “external property files”, and we refer to the file containing the run object itself as the “root file”. We refer to a property that can be stored in an external property file as an “externalizable property.” We refer to a property that *has* been stored in an external property file as an “external property.”

An external property file **SHALL** contain at least one external property; it **MAY** contain more than one. The format of an external property file is described in §4.

A SARIF consumer **SHALL** treat the contents of a property stored in an external property file exactly as if they had appeared inline in the root file as the value of the corresponding property of the run object. In particular, this means that if the resources property (§3.13.15) is externalized, its contents take precedence over a resource file for the language specified by tool.language (§3.16.8) that might be located by the resource file lookup procedure (§3.11.6.4).

#### Property definition

A run object **MAY** contain a property named externalPropertyFiles whose value is an object (§3.6). Each property name in this object **SHALL** equal the name of an externalizable property of the run object.

The following properties are externalizable:

* conversion
* files
* graphs
* invocations
* logicalLocations
* properties
* resources
* results

If the value of an externalizable property is an object, then if the property is externalized, it **SHALL** be stored in a single external property file. In that case, the value of the corresponding property in externalPropertyFiles **SHALL** be an externalPropertyFile object (§3.14) specifying the location of the external property file.

The following externalizable properties have values that are objects:

* conversion
* graphs
* properties
* resources

If the value of an externalizable property is an array, then if the property is externalized, it **SHALL** be stored in one or more external property files. In this case, the value of the corresponding property in externalPropertyFiles **SHALL** be an array of externalPropertyFile objects specifying the locations of those external property files.

The following externalizable properties have values that are arrays:

* files
* invocations
* logicalLocations
* results

EXAMPLE 1: In this example, the run.conversion property is stored in the file C:\logs\scantool.conversion.sarif-external-properties and run.results is divided into the files C:\logs\scantools.results-1.sarif-external-properties and C:\logs\scantools.results-2.sarif-external-properties.

{ # A run object.

"originalUriBaseIds": { # See §3.13.10.

"LOGSDIR": {

"uri": "file:///C:/logs/"

}

},

"externalPropertyFiles": {

"conversion": { # An externalPropertyFile object (§3.14).

"fileLocation": { # See §3.14.2.

"uri": "scantool.conversion.sarif-external-properties",

"uriBaseId": "LOGSDIR",

"fileIndex": 0

},

"instanceGuid": "11111111-1111-1111-1111-111111111111" # See §3.14.3.

},

"results": [

{

"fileLocation": {

"uri": "scantool.results-1.sarif-external-properties",

"uriBaseId": "LOGSDIR",

"fileIndex": 1

},

"instanceGuid": "22222222-2222-2222-2222-222222222222",

"itemCount": 10000

},

{

"fileLocation": {

"uri": "scantool.results-2.sarif-external-properties",

"uriBaseId": "LOGSDIR",

"fileIndex": 2

},

"instanceGuid": "333333333333-3333-3333-3333-333333333333",

"itemCount": 4277

}

]

}

...

}

If a property appears inline in the root file, its name **SHALL NOT** appear as one of the property names in externalPropertyFiles. Even if the property name erroneously appears in externalPropertyFiles, a SARIF consumer **SHALL** ignore the contents of the external property file.Since an external property file can contain multiple external properties, externalPropertyFile objects belonging to distinct properties **MAY** denote the same external property file. However, if an array-valued external property is divided among multiple external property files, the externalPropertyFile objects belonging to that property **SHALL** denote distinct external property files.

EXAMPLE 2: In this example, the run.conversion property and the run.properties property are stored in the same external property file.

{ # A run object.

"originalUriBaseIds": { # See §3.13.10.

"LOGSDIR": {

"uri": "file:///C:/logs/"

}

},

"externalPropertyFiles": {

"conversion": { # An externalPropertyFile object (see §3.14).

"fileLocation": { # See §3.14.2.

"uri": "scantool.sarif-external-properties",

"uriBaseId": "LOGSDIR",

"fileIndex": 0

},

"instanceGuid": "11111111-1111-1111-1111-111111111111" # See §3.14.3.

},

"properties": {

"fileLocation": {

"uri": "scantool.sarif-external-properties",

"uriBaseId": "LOGSDIR",

"fileIndex": 0

},

"instanceGuid": "11111111-1111-1111-1111-111111111111"

}

}

...

}

EXAMPLE 3: This example represents invalid SARIF because both elements of the array belonging to the run.results property denote the same external property file.

{ # A run object.

"originalUriBaseIds": { # See §3.13.10.

"LOGSDIR": {

"uri": "file:///C:/logs/"

}

},

"externalPropertyFiles": {

"results": [

{ # An externalPropertyFile object (see §3.14).

"fileLocation": {

"uri": "scantool.results.sarif-external-properties",

"uriBaseId": "LOGSDIR",

"fileIndex": 0

},

"instanceGuid": "22222222-2222-2222-2222-222222222222"

},

{ # INVALID: The two external property files are the same.

"fileLocation": {

"uri": "scantool.results.sarif-external-properties",

"uriBaseId": "LOGSDIR",

"fileIndex": 0

},

"instanceGuid": "22222222-2222-2222-2222-222222222222"

}

]

}

...

}

### id property

A run object **MAY** contain a property named id whose value is an runAutomationDetails object (§3.15) that describes this run.

For an example, see §3.15.1.

### aggregateIds property

A run object **MAY** contain a property named aggregateIds whose value is an array of zero or more unique (§3.7.2) runAutomationDetails objects (§3.15) each of which describes an aggregate of runs to which this run belongs.

If aggregateIds is absent, it **SHALL** default to an empty array.

For an example, see §3.15.1.

### baselineInstanceGuid property

A run object **MAY** contain a property named baselineInstanceGuid whose value is a GUID-valued string (§3.5.3) which **SHALL** equal the id.instanceGuid property (§3.13.3, §3.15.5) of some previous run.

NOTE: This ensures that only “similar” runs are compared.

If baselineInstanceGuid is present, the result.baselineState property (§3.22.20) of every result object (§3.22) in the containing run object **SHALL** be computed with respect to the run specified by baselineInstanceGuid.

### tool property

A run object **SHALL** contain a property named tool whose value is a tool object (§3.16) 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 zero or more invocation objects (§3.17) that describe the invocation of the analysis tool that was run.

If invocations is absent, it **SHALL** default to an empty array.

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 (§3.18) 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.

### versionControlProvenance property

A run object **MAY** contain a property named versionControlProvenance whose value is an array of zero or more unique (§3.7.2) versionControlDetails objects (§3.20). Each array entry specifies a revision in a repository containing files that were scanned during the run.

If versionControlProvenance is absent, it **SHALL** default to an empty array.

NOTE 1: This property allows an engineering system to reproduce a scan by retrieving the specified revision of the required files from of each repository before repeating the analysis run.

NOTE 2: This property is an array, rather than a single versionControlDetails object, to support scenarios where a tool scans files from multiple repositories in a single run.

NOTE 3: This specification refers to a container for a related set of files in a VCS as a “repository.” Different VCSs use different terms; for example, Visual Studio Team Services Version Control calls it a “team project”.

NOTE 4: This specification refers to a fixed revision of a set of files as a “revision”. Different VCSs use different terms; for example, Git calls it a “commit”.

EXAMPLE: In this example, an analysis tool has scanned files from one repository: the GitHub repository example/browser.

{ # A run object.

"versionControlProvenance": [

{ # A versionControlDetails object (§3.20).

"repositoryUri": "https://github.com/example/browser", # See §3.20.3.

"revisionId": "fd3fbae" # See §3.20.4.

"branch": "master" # See §3.20.5.

}

]

}

### originalUriBaseIds property

A run object **MAY** contain a property named originalUriBaseIds whose value is an object (§3.6) each of whose property names designates a URI base id (§3.4.4) and each of whose property values is a fileLocation object (§3.4) that specifies (in the manner described below) the absolute URI [[RFC3986](#RFC3986)] of that URI base id on the machine where the SARIF producer ran.

If the fileLocation object’s uri property (§3.4.3) is a relative reference, its uriBaseId property **SHALL** be present. Otherwise (that is, if uri is an absolute URI), uriBaseId **SHALL** be absent.

The values of the uriBaseId properties in the fileLocation objects in originalUriBaseIds **SHALL NOT** form a loop, in the sense described in the URI base id resolution procedure below.

This property allows SARIF consumers to resolve any relative references which appear in any fileLocation objects elsewhere 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.

A SARIF consumer **SHALL** use the following procedure to resolve a URI base id from the information in originalUriBaseIds:

NOTE 1: This procedure is part of an overall URI base id resolution procedure described in §3.4.4.

NOTE 2: In this procedure, we refer to the resolved URI value by the variable name resolvedUri.

1. Set resolvedUri to an empty string.
2. Fetch the fileLocation object whose property name within originalUriBaseIds is the value of uriBaseId. If there is no such property, the resolution procedure fails.
3. Prepend fileLocation.uri to resolvedUri.
4. If fileLocation.uri is an absolute URI, resolvedUri is the final resolved URI, and the procedure succeeds.  
     
   Otherwise:
5. If uriBaseId is absent, the resolution procedure fails.

NOTE 3: This would not occur in a valid SARIF file, but the file might not be valid.

1. If the value of uriBaseId has already been encountered during this resolution procedure (that is, if there is a loop in the sequence of URI base ids), the resolution procedure fails.

NOTE 4: This would not occur in a valid SARIF file, but the file might not be valid.

1. Otherwise (that is, if uriBaseId is present and its value has not previously been encountered during this resolution), return to Step 2.

EXAMPLE: In this example, the URI base id "SRCROOT" on the machine where the SARIF producer ran was "file:///C:/code/MyProject/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:\code\MyProject\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.

"originalUriBaseIds": {

"PROJECTROOT": {

"uri": "file:///C:/code/MyProject/"

},

"SRCROOT": {

"uri": " src/",

"uriBaseId": "PROJECTROOT"

}

},

"results": [

{ # A result object (§3.22).

"ruleId": "CA1001",

"locations": [

{ # A location object (§3.23).

"physicalLocation": { # See §3.23.3.

"fileLocation": { # A fileLocation object (§3.3).

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

"uriBaseId": "SRCROOT"

}

}

}

]

}

]

}

### files property

A run object **SHOULD** contain a property named files whose value is an array of zero or more unique (§3.7.2) file objects (§3.21) each of which represents a file relevant to the run.

If files is absent, it **SHALL** default to an empty array.

The array **SHOULD** contain elements representing at least those files in which results were detected, but it **MAY** contain elements 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 (§3.17.5, §3.22.22).

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": [

{

"fileLocation": {

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

},

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

"hashes": {

"sha-256": "b13ce2678a8807ba0765ab94a0ecd394f869bc81"

}

}

}

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”. Within the files array, a file object representing a nested file is linked to its parent *via* its parentIndex property (§3.21.3). For an example, see §3.21.3.

If a nested file appears in the files array, then the files array **SHALL** also contain elements describing each of its parents, up to and including the top-level file.

### logicalLocations property

Depending on the circumstances, a run object either **MAY** or **SHOULD** contain a property named logicalLocations whose value is an array of zero or more unique (§3.7.2) logicalLocation objects (§3.27) each of which represents a logical location relevant to one or more results detected during the run.

If logicalLocations is absent, it **SHALL** default to an empty array.

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

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

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”. . Within the logicalLocations array, a logicalLocation object representing a nested logical location is linked to its parent *via* its parentIndex property (§3.27.7).

If a nested logical location appears in the logicalLocations array, then the logicalLocations array **SHALL** also contain elements 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 array contains not only an element describing the class, but also elements describing its containing namespaces.

"logicalLocations": [

{

"name": "classC",

"fullyQualifiedName": "namespaceA::namespaceB::classC",

"kind": "type",

"parentIndex": 1

},

{

"name": "namespaceB",

"fullyQualifiedName": "namespaceA::namespaceB",

"kind": "namespace"

"parentIndex": 2

},

{

"fullyQualifiedName": "namespaceA",

"kind": "namespace"

}

]

NOTE: The detailed information in logicalLocations is useful, even though much of it is captured in location.fullyQualifiedLogicalName (§3.23.4), because it allows results management systems and other SARIF consumers 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.

### graphs property

A run object **MAY** contain a property named graphs whose value is an object (§3.6) each of whose property values is a graph object (§3.30), and each of whose property names equals the id property (§3.30.2) of that graph object. A graph object represents a directed graph: a network of nodes and directed edges that describes some aspect of the structure of the code (for example, a call graph).

A graph object defined at the run level **MAY** be referenced by a graphTraversal object (§3.33) defined in the graphTraversals property (§3.22.16) of any result object (§3.22) in the run.

### results property

A run object **SHALL** contain a property named results whose value is either null or an array of zero or more result objects (§3.22), each of which represents a single result detected in the course of the run.

NOTE: The results array is not defined to contain unique (§3.7.2) 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.

If the tool failed to start, and if the engineering system system responsible for running the tool synthesized a SARIF file to record the failure, then results **SHALL** be null. See §3.17.12, invocation.processStartFailureMessage, for more about this scenario.

If the tool started but failed to begin its analysis (for example, because its command line was invalid), then again results **SHALL** be null.

In all other circumstances, results **SHALL** be present and **SHALL** contain all results detected by the tool. If the tool did not detect any results, results **SHALL** be an empty array.

### resources property

A run object **MAY** contain a property named resources whose value is a resources object (§3.39). A resources object represents items that can be localized, such as resource strings and rule metadata.

### defaultFileEncoding property

A run object **MAY** contain a property named defaultFileEncoding whose value is a string that provides a default for the encoding property (§3.21.9) of any file object (§3.21) in run.files (§3.13.11) that refers to a text file. The string **SHALL** be one of the character set names specified in [[IANA-ENC](#IANA_ENC)]. The property value **SHALL** be case-insensitive.

If this property is absent, it **SHALL** be interpreted as meaning that there is no default file encoding. In that case, the encoding of any file object that does not contain an encoding property **SHALL** be taken to be unknown.

For an example, see §3.21.9.

### defaultSourceLanguage property

A run object **MAY** contain a property named defaultSourceLanguage whose value is a hierarchical string (§3.5.4) that provides a default value for the sourceLanguage property (§3.21.10) of any file object (§3.21) in run.files (§3.13.11) which refers to a text file that contains source code.

If defaultSourceLanguage is present, its value **SHOULD** conform to the conventions defined in §3.21.10.2.

If defaultSourceLanguage is absent, it **SHALL** be taken to mean that there is no default source language. In that case, the source language of any file object that does not contain a sourceLanguage property **SHALL** be taken to be unknown. In that case, a SARIF viewer **MAY** use any method or heuristic to determine the source language of each file, for example by examining the file’s file name extension or MIME type, or by prompting the user.

### newlineSequences property

A run object **MAY** contain a property named newlineSequences whose value is an array of one or more unique (§3.7.2) strings, each of which specifies a character sequence that the tool treated as a line break during this run.

If this property is absent, it **SHALL** default to the array [ "\r\n", "\n" ].

The order of the elements in the array is significant. It **SHALL** mean that at potential line breaks, the tool “greedily” attempted to match each element of the array in order.

EXAMPLE 1: If newlineSequences has the value [ "\r\n", "\r", "\n" ], the character sequence "\r\n" counts as one line break, not two.

NOTE: This property is useful for SARIF consumers that are sensitive to the value of the line number properties startLine (§3.25.5) and endLine (§3.25.7) in region objects (§3.25). It ensures that the consumer counts lines in the same way as the producer. A SARIF viewer might use this property when highlighting a region to ensure that it highlights the correct lines. More critically, a tool that applies fixes (see §3.42), especially one that applies them automatically, can use this property to ensure that it inserts and removes file content on the correct lines.

EXAMPLE 2: In this example, the SARIF producer accepts the Unicode characters NEXT LINE (U+0085) and LINE SEPARATOR (U+2028) as line separators in addition to the usual values.

{ # A run object (§3.13).

...

"newlineSequences": [ "\r\n", "\n", "\u0085", "\u2028" ],

...

}

### columnKind property

If a SARIF producer processes text files, the run object **SHALL** contain a property named columnKind whose value is a string that specifies the unit in which the analysis tool measures columns.

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

* "utf16CodeUnits": Each UTF-16 code unit is considered to occupy one column. This means that a surrogate pair is considered to occupy two columns.
* "unicodeCodePoints": Each Unicode code point (abstract character) is considered to occupy one column. This means that even a character that is represented in UTF-16 by a surrogate pair is considered to occupy one column.

If the SARIF producer does not process text files, columnKind **SHALL** be absent.

If the SARIF producer processes text files and columnKind is absent, it **SHALL** default to unicodeCodePoints.

If a SARIF consumer uses a column measurement unit other than that specified by columnKind, and if the consumer is required to interact with the file contents (for example, by displaying the file in an editor and highlighting a region), the consumer **SHALL** recompute column numbers in its (the consumer’s) native measurement unit.

### 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 (§3.11.3) in the run. If this property is absent, it **SHALL** default to "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 text messages in GFM, see §3.11.3.2.

### redactionToken property

If the value of any redactable property (§3.5.2) in the run has been redacted, the run object **SHALL** contain a property named redactionToken whose value is the string used to replace the redacted text. If no text in the run has been redacted, the redactionToken property **SHALL** be absent.

The value of redactionToken **SHOULD** be the string "[REDACTED]". If for any reason a different value is used, it **MAY** be any readily identifiable string. An example of a situation where a SARIF producer might choose a different redaction token is if the string "[REDACTED]" occurs in the value of any redactable property in the run.

EXAMPLE 1: In this example, the leading portion of a full path name has been redacted from the redactable property invocation.commandLine to avoid revealing information about the machine.

{ # A run object (§3.13).

"redactionToken": "[REDACTED]",

"invocation": {

"commandLine": "SourceScanner --input [REDACTED]/src/ui"

}

...

}

## externaPropertyFile object

### General

An externalPropertyFile object contains information that enables a SARIF consumer to locate the external property file (see §3.13.2) that contains the value of an external property.

### fileLocation property

An externalPropertyFile object **SHALL** contain a property named fileLocation whose value is a fileLocation object (§3.4) that specifies the location of the external property file.

If the fileLocation object’s uri property (§3.4.3) specifies a relative reference and its uriBaseId property (§3.4.4) is absent, then uri **SHALL** be interpreted relative to the location of the root file.

### instanceGuid property

An externalPropertyFile object **SHALL** contain a property named instanceGuid whose value is a GUID-valued string (§3.5.3) which provides a unique, stable identifier for the external property file.

### itemCount property

If an externalPropertyFile object specifies an external property file that contains all or a portion of an array-valued property, it **MAY** contain a property named itemCount whose value is an integer greater than or equal to 1 that specifies the number of items in the externalized property array in that file. If the externalPropertyFile object specifies an external property file that contains an object-valued property, itemCount **SHALL** be absent.

NOTE: This information is useful to a SARIF consumer that needs to locate the item at a specified array index in an externalized array-valued property. Without this information, the consumer would have to open in turn each external property file belonging to that property, counting the number of array elements in each, until it reached the file containing the desired element.

EXAMPLE: In EXAMPLE 1 in §3.13.2.2, the array-valued property results is divided into two files, the first containing 10,000 elements and the second containing 4,277 elements. A SARIF consumer that needs to access element 12,000 knows immediately that it is contained in the second file, at index 2,000.

## runAutomationDetails object

### General

A runAutomationDetails object contains information that specifies its containing run object’s (§3.13) identity and role within an engineering system.

EXAMPLE: In this example, a run contains the results from one nightly execution of a single security tool over a specified set of binaries. run.id describe the containing run. Its instanceId and instanceGuid properties both identify the the run; the former in human-readable form, the latter in a form that might be more useful in an engineering system’s database. Its correlationGuid property specifies the set of runs identified by *all but the last component* of instanceId’s hierarchical string; that is, it identifies the set of runs "Nightly CredScan run for sarif-sdk/master/x86/debug".

The run in this example is part of an aggregate of runs which together comprise the nightly execution of the engineering system’s full suite of security tools. run.aggregateIds[0] describes that aggregate. Its instanceId and instanceGuid properties both identify the aggregate. Its correlationGuid property specifies the collection of such aggregates identified by *all but the last component* of instanceId’s hierarchical string; this is, it identifies the collection of aggregates "Nightly security tools run for sarif-sdk/master/x86/debug".

{ # A run object (§3.13).

"id": { # See §3.13.3.

"description": {

"text": "This is the {0} nightly run of the Credentaial Scanner tool on

all product binaries in the '{1}' branch of the '{2}' repo. The

scanned binaries are architecture '{3}' and build type '{4}'.",

"arguments": [

"October 10, 2018",

"master",

"sarif-sdk",

"x86",

"debug"

]

},

"instanceId":

"Nightly CredScan run for sarif-sdk/master/x86/debug/2018-10-05",

"instanceGuid": "11111111-...",

"correlationGuid": "22222222-...."

},

"aggregateIds": [ # See §3.13.4.

{

"instanceId":

"Nightly security tools run for sarif-sdk/master/x86/debug/2018-10-05"

"instanceGuid": "33333333-...",

"correlationGuid": "44444444-...."

}

]

}

### Constraints

At least one of the instanceId (§3.15.4), instanceGuid (§3.15.5), or correlationGuid (§3.15.6) properties **SHALL** be present.

### description property

An runAutomationDetails object **MAY** contain a property named description whose value is a message object (§3.11) that describes the role played within the engineering system by this object’s containing run object (§3.13).

### instanceId property

An runAutomationDetails object **MAY** contain a property named instanceId whose value is a hierarchical string (§3.5.4) that uniquely identifies this object’s containing run (§3.13) object within the engineering system.

A result management system or other components of the engineering system **MAY** use run.id.instanceId to associate the information in the log with additional information not provided by the analysis tool that produced it.

An engineering system **MAY** define any number of components and interpret them in any way desired.

NOTE: The intent is to use the components of instanceId to group results from similar runs, such as “all nightly Credential Scanner runs.” A SARIF viewer might display a set of runs in a tree view, grouped by the components of instanceId.

EXAMPLE 1: A run whose instanceId is "My Nightly Run/Debug/x64/2018-10-10" belongs to the category "My Nightly Run/Debug/x64". Presumably, this is the run from October 10, 2018.

The trailing component of instanceId **MAY** be empty; note that the grammar for a hierarchical identifier (§3.5.4.1) permits any component to be empty. This **SHALL** be taken to signify that the run belongs to the specified category, but that the run itself has no unique identifier.

EXAMPLE 2: A run whose instanceId is "My Nightly Run/Debug/x64/" belongs to the category "My Nightly Run/Debug/x64" but is not distinguished from other runs in that category.

instanceId **MAY** consist of a single component. This **SHALL** be taken to specify a unique identifier for the run, withough specifying any category that the run belongs to.

EXAMPLE 3: A run whose instanceId is "My Nightly Run Debug x64 2018-10-10" has a unique identifier but cannot be inferred to belong to any category.

### instanceGuid property

An runAutomationDetails object **MAY** contain a property named instanceGuid whose value is a GUID-valued string (§3.5.3) that provides a unique, stable identifier for this object’s containing run object.

A result management system or other components of the engineering system **MAY** use run.id.instanceGuid to associate the information in the log with additional information not provided by the analysis tool that produced it.

### correlationGuid property

An runAutomationDetails object **MAY** contain a property named correlationGuid whose value is a GUID-valued string (§3.5.3) which is shared by all such runs of the same type, and differs between any two runs of different types.

If instanceId (§3.15.4) is present, correlationGuid **SHALL** identify the category of runs specified by all but the last hierarchical component of instanceId.

NOTE: Consider an engineering system that allows engineers to define “build definitions”, and that assigns a GUID to each build definition. In such a system, the build definition’s GUID could serve as run.id.correlationGuid. It would be the same for all runs that produced by the same build definition, and different between any two runs produced by different build definitions.

## 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 §3.16.2

"fullName": "CodeScanner 1.1, Developer Preview (en-US)", # see §3.16.3

"semanticVersion": "1.1.2-beta.12", # see §3.16.4

"version": "1.1.2b12, # see §3.16.5

"dottedQuadFileVersion": "1.1.1502.2" # see §3.16.6

}

### 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 **MAY** contain a property named semanticVersion whose value is a string containing the tool version in a format that conforms to the syntax and semantics specified by [[SEMVER](#SEMVER)].

EXAMPLE 1:

"tool": {

"semanticVersion": "1.1.2-beta.12"

}

NOTE 1: Semantic versions are 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.

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.

### dottedQuadFileVersion property

If the operating system on which the tool runs provides a value for the file version of the tool's primary executable file, and if that value logically consists of an ordered set of four non-negative integers, then the tool object **MAY** contain a property named dottedQuadFileVersion whose value is a string representation of that file version in this syntax:

dottedQuadFileVersion = non negative integer, 3\*(".", non negative integer);

where the non negative integers follow the logical order of the components of the file version.

If the operating system does not provide such a value, the dottedQuadFileVersion property **SHALL** be absent.

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

### downloadUri property

A tool object **MAY** contain a property named downloadUri whose value is a string containing the absolute URI [[RFC3986](#RFC3986)] from which this version of the tool can be downloaded.

### 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, it **SHALL** default to "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 (§3.11.7) and richText (§3.11.8) properties of any message object (§3.11) in the containing run object (§3.13).
2. The language of any embedded resources (§3.11.6) contained in the resources property (§3.13.15) of the containing run object.

### resourceLocation property

If a SARIF producer provides external resources (§3.11.6) for languages other than the tool’s declared language (§3.16.8), the tool object **SHALL** contain a property named resourceLocation whose value is a fileLocation object (§3.3) 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 (§3.4.3) specifies a relative reference, then its uriBaseId property (§3.4.4) **SHOULD** be present, and the run object’s originalUriBaseIds property (§3.13.10) **SHOULD** contain a property corresponding to 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 (§3.13).

"tool": {

"name": "SecurityScanner",

"version": "2.0.1",

"resourceLocation": { # A fileLocation object (§3.3).

"uri": "resources",

"uriBaseId": "TOOLINSTALLDIR"

}

},

"originalUriBaseIds": { # See §3.13.10.

"TOOLINSTALLDIR": {

"uri": "file:///C:/Program%20Files/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 (§3.13).

"tool": {

"name": "SecurityScanner",

"version": "2.0.1",

"resourceLocation": { # A fileLocation object (§3.3).

"uri": "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.

## 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.

The commandLine property is redactable (§3.5.2) because it might contain information which it is not appropriate to disclose, such as passwords, tokens, database connection strings, or in some circumstances even the fully qualified path to the tool's executable or script file.

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 commandLine might contain the redacted string

[REDACTED]\DbScanner.exe /connectionString=[REDACTED] /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 **SHALL 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 either null or an array of zero or more strings, containing in order the command line arguments passed to the tool from the operating system.

If arguments is absent, it **SHALL** default to null.

An empty array **SHALL** mean that the tool was invoked with no command line arguments. null **SHALL** mean that the command line arguments, if any, are not known.

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 (§3.17.2) 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 either null or an array of zero or more unique (§3.7.2) fileLocation objects (§3.4), each of which represents a response file specified on the tool's command line.

If responseFiles is absent, it **SHALL** default to null.

An empty array **SHALL** mean that the tool was invoked with no command line arguments that specified response files. null **SHALL** mean that it is not known whether any command line arguments specified a response file.

A SARIF producer **MAY** embed the contents of a response file in the SARIF log file by mentioning the response file in run.files (§3.13.11) and providing a value for file.contents (§3.21.8).

EXAMPLE:

{ # An invocation object.

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

"responseFiles": [

{ # A fileLocation object (§3.4)..

"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 zero or more unique (§3.7.2) attachment objects (§3.18), each of which describes a file relevant to the invocation of the tool. The array elements **SHALL** refer to distinct files.

NOTE: Typically, these would be files specified on the tool’s command line, and therefore mentioned in the commandLine property (§3.17.2) or the arguments property (§3.17.3), if present. They might also be files implicitly consumed by the tool, such as a configuration file.

If attachments is absent, it **SHALL** default to an empty array.

EXAMPLE 1: In this example, the two attachment objects refer to distinct image files. The rectangles of interest happen to be the same in each file.

{ # An invocation object.

"attachments": [

{ # An attachment object (§3.18)

"fileLocation": { # See §3.18.3

"uri": "images/screenshot-before.png",

"uriBaseId": "IMAGE\_DIR"

},

"rectangles": [ # See §3.18.5.

{ # A rectangle object (§3.26).

"top": 250,

"left": 100,

"bottom": 350,

"right": 150

}

]

},

{

"fileLocation": {

"uri": "images/screenshot-after.png",

"uriBaseId": "IMAGE\_DIR"

},

"rectangles": [

{

"top": 250,

"left": 100,

"bottom": 350,

"right": 150

}

]

}

]

}

EXAMPLE 2: This example represents invalid SARIF because the two attachment objects refer to the same file, images/screenshot.png. It is invalid even though the attachment objects are distinguished by their rectangles properties.

{ # An invocation object.

"attachments": [

{ # An attachment object (§3.18)

"fileLocation": { # See §3.18.3

"uri": "images/screenshot.png",

"uriBaseId": "IMAGE\_DIR"

},

"rectangles": [ # See §3.18.5.

{ # A rectangle object (§3.26).

"top": 250,

"left": 100,

"bottom": 350,

"right": 150

}

]

},

{

"fileLocation": {

"uri": "images/screenshot.png", # INVALID: the same file...

"uriBaseId": "IMAGE\_DIR"

},

"rectangles": [

{ # ... with a different rectangle.

"top": 750,

"left": 100,

"bottom": 850,

"right": 150

}

]

}

]

}

### startTimeUtc property

An invocation object **MAY** contain a property named startTimeUtc whose value is a string specifying the UTC date and time at which thetool’s execution started. The string **SHALL** be in the format specified in §3.9.

### endTimeUtc property

An invocation object **MAY** contain a property named endTimeUtc whose value is a string specifying the UTC date and time at which the tool’s execution ended. The string **SHALL** be in the format specified in §3.9.

### 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 §3.17.9.

### 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 §3.17.11.

### 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, it **SHALL** default to false if the exitCode property (§3.17.8) is present and has a non-zero value; otherwise it **SHALL** default to 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.

### executableLocation property

An invocation object **MAY** contain a property named executableLocation whose value is a fileLocation object (§3.4) specifying the location of the tool's executable file.

NOTE 1: This property is defined in the invocation object rather than in the tool object (§3.16) because the identical tool might be invoked from different paths on different machines.

NOTE 2: This property might duplicate information in the commandLine property (§3.17.2). 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 fileLocation object (§3.4) specifying 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 corresponding property value **SHALL** be an empty string.

NOTE 1: Environment variable names and values are likely to reveal highly sensitive information. For example, on a machine running Microsoft Windows ®, 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 Microsoft 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 (§3.45). 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 (§3.45.7) is "error" **SHALL** mean that the run failed.

If toolNotifications is absent, it **SHALL** default to an empty array.

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 (§3.45). 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 (§3.45.7) is "error" **SHALL** mean that the run failed.

If configurationNotifications is absent, it **SHALL** default to an empty array.

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 fileLocation objects (§3.4) 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".

A SARIF producer **MAY** embed the stream contents in the log file by mentioning the corresponding file in run.files (§3.13.11) and providing a value for file.contents (§3.21.8).

## attachment object

### General

An attachment object describes a file relevant to the invocation of a tool (see §3.17.5) or to the detection of a result (see §3.22.22).

A SARIF producer **MAY** embed the contents of an attachment in the log file by mentioning the attachment file in run.files (§3.13.11) and providing a value for file.contents (§3.21.8).

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

{ # A run object (§3.13).

"invocations": [ # See §3.13.7.

{ # An invocation object (§3.17).

...

"attachments": [ # See §3.17.5.

{ # An attachment object.

"description": { # See §3.18.2.

"text": "Configuration file"

},

"fileLocation": { # See §3.18.3.

"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 (§3.22).

...

"attachments": [ # See §3.22.22.

{ # An attachment object.

"description": { # See §3.18.2.

"text": "Screen shot"

},

"fileLocation": { # See §3.18.3.

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

}

}

]

}

### description property

An attachment object **SHOULD** contain a property named description whose value is a message object (§3.11) describing the role played by the attachment.

### fileLocation property

An attachment object **SHALL** contain a property named fileLocation whose value is a fileLocation object (§3.3) that specifies the location of the attachment file.

### regions property

An attachment object **MAY** contain a property named regions whose value is an array of zero or more unique (§3.7.2) region objects (§3.25), each of which **SHALL** specify a region of interest within the attachment, and **SHOULD** contain a message property (§3.25.14) so a user can understand its relevance.

If regions is absent, it **SHALL** default to an empty array.

### rectangles property

An attachment object **MAY** contain a property named rectangles whose value is an array of zero or more unique (§3.7.2) rectangle objects (§3.26). If the attachment is an image file (for example .png or .svg), each rectangle object **SHALL** specify an area of interest within the image, and **SHOULD** contain a message property (§3.26.3) so a user can understand its relevance.

If the attachment is not an image file, and rectangles is present, its value **SHALL** be an empty array.

If rectangles is absent, it **SHALL** default to an empty array.

## 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 §3.19.2

"name": "SARIF SDK Multitool",

},

# see §3.19.3

"invocation":

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

"analysisToolLogFileLocation": { # see §3.19.4

"uri": "northwind.log",

"uriBaseId": "$LOG\_DIR$"

}

},

"results": [

...

]

}

]

}

### tool property

A conversion object **SHALL** contain a property named tool whose value is a tool object (§3.16) that describes the converter.

### invocation property

A conversion object **MAY** contain a property named invocation whose value is an invocation object (§3.17) that describes the invocation of the converter.

### analysisToolLogFiles property

Some analysis tools produce one or more output files that describe the analysis run as a whole; we refer to these as “per-run” files. Some tools produce one or more output files for each result; we refer to these as “per-result” files. Some tools produce both per-run and per-result files.

A conversion object **MAY** contain a property named analysisToolLogFiles whose value is an array of zero or more unique (§3.7.2) fileLocation objects (§3.3) that specify the locations of the per-run files.

If the analysis tool did not produce any per-run files, and analysisToolLogFiles is present, its value **SHALL** be an empty array.

If analysisToolLogFiles is absent, it **SHALL** default to an empty array.

Per-result files are handled by the resultProvenance.conversionSources property (§3.38.7).

## versionControlDetails object

### General

A versionControlDetails object specifies the information necessary to retrieve from a version control system (VCS) the correct revision of the files that were scanned during the containing run (§3.13).

For an example, see §3.13.9.

### Constraints

A versionControlDetails object **SHALL** contain sufficient information to uniquely and permanently identify the revision of the files that were scanned.

NOTE: The required set of properties depends on the VCS and on the engineering system within which it is used. Consider Git as an example. The revisionId property (containing a commit id) would suffice. The branch property (§3.20.5) might not suffice because a Git branch is a pointer to the latest commit along a line of development; however, branch together with asOfTimeUtc (§3.20.7) might suffice (although that is not an idiomatic use of Git). Similarly, revisionTag (§3.20.6) might not suffice because a Git tag can be removed, but if the engineering system guaranteed that certain tags (such as those specifying public releases) were stable, then revisionTag might suffice.

### repositoryUri property

A versionControlDetails object **SHALL** contain a property named repositoryUri whose value is a string containing an absolute URI [[RFC3986](#RFC3986)] that specifies the location of the repository containing the scanned files.

### revisionId property

A versionControlDetails object **SHOULD** contain a property named revisionId whose value is a string that uniquely and permanently identifies the appropriate revision of the scanned files.

### branch property

A versionControlDetails object **MAY** contain a property named branch whose value is a string containing the name of a branch containing the correct revision of the scanned files.

### revisionTag property

A versionControlDetails object **MAY** contain a property named revisionTag whose value is a string containing a tag that has been applied to the revision in the VCS.

NOTE 1: This specification refers to an identifier for a revision in a VCS as a “tag”. Different VCSs use different terms; for example, Visual Studio Team Services Version Control calls it a “label”.

NOTE 2: Although VCSs generally allow a revision to have more than one tag, the revisionTag property is not an array. The purpose of revisionTag is to aid in identifying a revision so that a scan can be reproduced, not to exhaustively describe the revision.

### asOfTimeUtc property

A versionControlDetails object **MAY** contain a property named asOfTimeUtc whose value is a string specifying a UTC date and time that can be used to synchronize an enlistment to the state of the repository as of that time. The string **SHALL** be in the format specified in §3.9.

NOTE: In some VCSs, the “synchronize by date” feature requires the time to be expressed in the server’s time zone. In such a case, the SARIF producer would need to know the server’s time zone to correctly populate asOfTimeUtc.

### mappedTo property

A versionControlDetails object **MAY** contain a property named mappedTo whose value is a fileLocation object (§3.4) that specifies the location in the local file system to which the root of the repository was mapped at the time of the analysis.

This property makes it possible to map any fileLocation to the repository, if any, to which the file belongs. The mapping algorithm **SHALL** be as follows, or any algorithm with equivalent effect (a clarifying example follows):

1. Resolve the fileLocation as far as possible using the procedure specified in §3.13.10. Denote the resolved fileLocation by f.
2. For every versionControlDetails object vcd in the versionControlProvenance property (§3.13.9) of the containing run object (§3.13), resolve the fileLocation object specified by vcd.mappedTo, again using the procedure specified in §3.13.10. Denote each such resolved fileLocation object by v.
3. Let S be the set of all versionControlDetails objects vcd for which v.uriBaseId equals f.uriBaseId and v.uri is a prefix of f.uri.
4. If S is the empty set, then the file specified by fileLocation does not belong to any repository.
5. Otherwise, then the file specified by fileLocation belongs to the repository specified by the member of S with the longest v.uri.

EXAMPLE: This example illustrates the mapping algorithm. Consider this SARIF file:

{

"originalUriBaseIds": {

"HOME": {

"uri": "file:///home/user"

}

"PACKAGE\_ROOT": {

"uri": "package",

"uriBaseId": "HOME"

},

},

"versionControlProvenance": [

{

"repositoryUri": "https://github.com/example-corp/package",

"revisionId": "b87c4e9"

"mappedTo": {

"uriBaseId": "PACKAGE\_ROOT"

},

},

{

"repositoryUri": "https://github.com/example-corp/plugin1",

"revisionId": "cafdac7"

"mappedTo": {

"uriBaseId": "PACKAGE\_ROOT"

"uri": "plugin1",

},

{

"repositoryUri": "https://github.com/example-corp/plugin2",

"revisionId": "d0dc2c0"

"mappedTo": {

"uriBaseId": "PACKAGE\_ROOT"

"uri": "plugin2",

}

],

"results": [

{

"ruleId": "CA1000",

"locations": [

{

"physicalLocation": {

"fileLocation": {

"uri": "plugin1/x.c",

"uriBaseId": "PACKAGE\_ROOT"

}

}

}

]

}

]

}

The object is to determine to which repository, if any, the file plugin1/x.c specified by the result location belongs. The algorithm proceeds as follows, using a simplified notation (*uriBaseId*, *uri*) to denote a fileLocation:

1. Use the information in originalUriBaseIds and the procedure specified in §3.13.10 to calculate the “resolved the file location” f:  
     
   (PACKAGE\_ROOT, plugin1/x.c) → (HOME, package/plugin1/x.c) → (null, file:///home/user/package/plugin1/x.c).
2. In the same way, calculate the resolved file location v from the mappedTo property of each element vcd of the versionControlProvenance array:
   1. (PACKAGE\_ROOT, null) → (HOME, package) → (null, file:///home/user/package)
   2. (PACKAGE\_ROOT, plugin1) → (HOME, package/plugin1) → (null, file:///home/user/package/plugin1)
   3. (PACKAGE\_ROOT, plugin2) → (HOME, package/plugin2) → (null, file:///home/user/package/plugin2)
3. The set of vcd for which v.uriBaseId equals f.uriBaseId (which is null) and for which v.uri is a *prefix* of f.uri (which is file:///home/user/package/plugin1/x.c) contains the objects at indices 0 and 1. It does not contain the object at index 2 because file:///home/user/package/plugin2 is not a prefix of file:///home/user/package/plugin1/x.c.
4. The set is not empty (it contains indices 0 and 1).
5. The member of the set for with the longest v.uri is the object at index 1, because file:///home/user/package/plugin1 is longer than file:///home/user/package.

Therefore, the specified file belongs to the repository specified by the versionControlDetails object at index 1, namely https://github.com/example-corp/plugin1.

## 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 (§3.3).

If the file object represents a top-level file, then fileLocation **SHALL** be present.

If the file object represents a nested file whose location relative to its parent can be expressed only by means of a path, then fileLocation **SHALL** be present, and the value of its uri property **SHALL** be a relative reference [[RFC3986](#RFC3986)] 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 (§3.21.4) 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 relative reference expressing the path of the nested file within the parent.

For an example, see §3.21.3.

### parentIndex property

If this file object represents a nested file, then it **SHALL** contain a property named parentIndex whose value is a non-negative integer that specifies the index of the parent file's file object within the run.files (§3.13.11) array. If this property is absent, it **SHALL** default to -1.

If this file object represents a top-level file, then parentIndex **SHALL** be absent.

NOTE: parentIndex 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.

EXAMPLE: This example demonstrates two levels of file nesting. The top-level file is a ZIP archive represented by the file object at index 0 in the files array. The archive contains a word processing document at the specified absolute path from its root; the document is represented by the file object at index 1. Finally, the document contains an embedded media object of the specified length at the specified offset from its beginning; the media object is represented by the file object at index 2. The media object’s parentIndex property refers to its parent document; the document’s parentIndex property refers to its parent ZIP archive, and the ZIP archive does not have a parentIndex property.

"files": [

{

"fileLocation": {

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

},

"mimeType": "application/zip",

},

{

"fileLocation": {

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

},

"mimeType":

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

"parentIndex": 0

},

{

"offset": 17522,

"length": 4050,

"mimeType": "application/x-contoso-animation",

"parentIndex": 1

}

}

### 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 (§3.21.2) 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.

### roles property

A file object **MAY** contain a property named roles whose value is an array of zero or more unique (§3.7.2) strings, each of which specifies a role that this file played in the analysis.

If roles is absent, it **SHALL** default to an empty array.

Each array element **SHALL** have one of the following values, with the specified meanings:

* "analysisTarget": The analysis tool was instructed to scan this file.
* "attachment": The file is an attachment mentioned in invocation.attachments (§3.17.5) or result.attachments (§3.22.22).
* "directory": The file object represents a directory (a container for other files and directories) rather than a file.

NOTE 1: URIs do not represent “directories” in the file system sense. Even if the URI https://www.example.com/dir/file addresses a resource, the URI https://www.example.com/dir might also address a resource. Nonetheless, if the analysis tool knows that https://www.example.com/dir is not itself a resource, but only a prefix for other URIs that *are* resources, it is appropriate for the tool to mark https://www.example.com/dir with the "directory" role.

* "responseFile": The file contains command line arguments to a program, as specified in invocation.responseFiles (§3.17.4).
* "resultFile": A result was detected in this file.
* "standardStream": The file contains the contents of one of the standard input or output streams, as specified in invocation.stdin, invocation.stdout, invocation.stderr, or invocation.stdoutStderr (§3.17.22).
* "traceFile": The analysis tool traced through this file while executing or simulating the execution of the code under test.

The following role values denote files that have changed since the baseline run. If baselineInstanceGuid (§3.13.5) is present on the containing run object (§3.13), its value **SHALL** specify the baseline run. If any of these role values are present but baselineInstanceGuid is absent, the engineering system **SHALL** provide out of band information that determines the baseline run.

* "unmodifiedFile": The file has not been modified since the baseline run.
* "modifiedFile": The file was modified after the baseline run.
* "addedFile": The file was added after the baseline run.
* "deletedFile": The file was deleted after the baseline run.
* "renamedFile": The file was renamed after the baseline run. In this case, the file object specifies the new name.
* "uncontrolledFile": The file is not under version control.

NOTE 2: The information conveyed by these values could be extracted from a VCS. These properties exist so SARIF consumers can have this information without needing access to the VCS.

### 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.

### contents property

A file object **MAY** contain a property named contents whose value is a fileContent object (§3.3) representing the entire contents of the file.

### encoding property

If a file object represents a text file, it **MAY** contain a property named encoding whose value is a string that specifies the file’s text encoding. The string **SHALL** be one of the character set names specified in [[IANA-ENC](#IANA_ENC)]. The property value **SHALL** be case-insensitive.

If the file object represents a text file and this property is absent, it **SHALL** default to the value of the defaultFileEncoding property (§3.13.16) of the containing run object (§3.13), if that property is present; otherwise, the file’s encoding **SHALL** be taken to be unknown.

If the file object represents a binary file, the encoding property **SHALL** be absent.

EXAMPLE: In this example, the encoding of output.txt is UTF-16BE (obtained from the default), but the encoding of data.txt is UTF-16LE:

{ # A run object (§3.13)

"defaultFileEncoding": "UTF-16BE", # See §3.13.16.

"files": [ # See §3.13.11.

{

"fileLocation": {

"uri": "output.txt"

}

# encoding property omitted

},

{

"fileLocation": {

"uri": "data.txt"

},

"encoding": "UTF-16LE"

}

]

}

### sourceLanguage property

#### General

If a file object represents a text file that contains source code, it **MAY** contain a property named sourceLanguage whose value is a hierarchical string (§3.5.4) that specifies the programming language in which the source code is written. If the file object does not represent a text file containing source code, sourceLanguage **SHALL** be absent.

For the remainder of this section, we assume that the file object represents a text file that contains source code.

NOTE 1: This property is intended to help SARIF viewers to render code snippets (§3.25.13) with appropriate syntax coloring.

If the file contains source code in a mix of languages, and if it is possible to identify one of those languages as the “primary” language of the file, then sourceLanguage **SHALL** specify that language.

NOTE 2: Typically, this is the language implied by the file name extension.

EXAMPLE: In an HTML file that contains embedded JavaScript, sourceLanguage would be "html".

If it is not possible to identify a primary language, sourceLanguage **MAY** specify any language used in the file, or it **MAY** be absent.

NOTE 3: In either case, it is possible to specify a source language for any region by using region.sourceLanguage (see §3.25.15).

If sourceLanguage is absent, it **SHALL** default to the value of the defaultSourceLanguage property (§3.13.17) of the containing run object (§3.13). If both file.sourceLanguage and run.defaultSourceLanguage are absent, the file’s source language **SHALL** be taken to be unknown. In that case, a SARIF viewer **MAY** use any method or heuristic to determine the file’s source language, for example, by examining its file name extension or MIME type, or by prompting the user.

#### Source language identifier conventions and practices

To maximize interoperability, SARIF producers and consumers **SHOULD** conform to the following conventions and practices with respect to the value of this property:

* Producers:
  + Use only lower-case letters, and numbers (for example, "c" rather than "C").
  + Spell out symbols (for example, "csharp" rather than "c#").
  + To denote a language variant, use the hierarchical string mechanism (for example, "c#/7").
  + Do not abbreviate (for example, "visualbasic"™ rather than "vb").
* Consumers
  + Accept source language identifiers that conform to the above producer conventions.
  + In addition, accept a variety of common industry forms, for example, {"cplusplus", "c++", "cpp"}, or {"javascript", "js"}.
  + Compare source language identifiers case-insensitively.

[Appendix I](#AppendixSourceLanguage), “Sample sourceLanguage values,” provides sample values for common programming languages.

### hashes property

A file object **MAY** contain a property named hashes whose value is a non-empty object (§3.6) each of whose property names specifies the name of a hash function, and each of whose property values represents the value produced by that hash function.

EXAMPLE: In this example, each of the hash functions SHA-256 and SHA-512 were used to compute hash values for the file.

{ # A file object.

"hashes": {

"sha-256": "...",

"sha-512": "..."

}

}

To maximize interoperability, the property names **SHOULD** appear in the IANA registry of hash function textual names [[IANA-HASH](#IANA_HASH)]. SARIF consumers that need to verify hash values **SHOULD** be able to compute any hash function whose name appears in the IANA registry.

The object **SHOULD** contain a property named "sha-256". SARIF consumers that need to verify hash values **SHALL** be able to compute a SHA-256 hash.

The object **MAY** contain properties whose names do not appear in the IANA registry, but at the expense of interoperability. A SARIF consumer **MAY** implement any hash function, but it does not have to implement any hash function that does not appear in the IANA registry.

If the hash function is one whose name appears in the IANA registry, the property name **SHALL** equal the name as it appears in the registry (for example, "sha-256" rather than "sha256"); otherwise the property name **MAY** be any suitable name, but it **SHALL NOT** equal any name defined in the IANA registry.

SARIF consumers **SHALL** treat the property name as case insensitive (even when comparing to hash function names in the IANA registry).

Each property value **SHALL** be a string representation of the hash digest of the file, computed by the hash function specified by the property name. Numeric hash values **SHALL** be represented in hexadecimal. The hexadecimal string representation **SHALL NOT** include a hexadecimal prefix such as "0x" or a suffix such as "h". A SARIF consumer **SHALL** treat a hexadecimal string value as case insensitive.

NOTE 1: The value is represented as a string because hash values are typically represented in hexadecimal notation, and JSON integer values must be decimal.

NOTE 2: 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 a set 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 hash function. Compliance systems, for example, will favor the use of more secure hash functions (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 function (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 hash functions required by those systems. Analysis tools that are configurable to produce hashes with a variety of commonly used hash functions will interoperate most easily with such systems.

### lastModifiedTimeUtc property

A file object **MAY** contain a property named lastModifiedTimeUtc whose value is a string specifying the UTC date and time at which the file was most recently modified. The string **SHALL** be in the format specified in §3.9.

NOTE: In scenarios where a tool has analyzed files on a network file share or on a local disk, an engineering system might use this property, rather than hashes (§3.21.11), as the most lightweight mechanism to determine whether the analysis needs to be repeated.

## result object

### General

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

### Distinguishing logically identical from logically distinct results

Successive runs of the same tool, or even runs of different tools, might detect the same condition in the code. When two result objects represent the same condition, we say that the results are “logically identical;” when they represent different conditions, we say that the results are “logically distinct.” Two results can be logically identical even if the result objects are not identical. For example, if code is inserted into the file between runs, the same condition might be reported on two different lines.

To avoid reporting the same condition repeatedly, result management systems typically group results into equivalence classes such that results in any one class are logically identical and results in different classes are logically distinct.

Some result management systems do this by calculating a “fingerprint” for each result and considering results with the same fingerprint to be logically identical. A fingerprint is calculated from information contained in the result and might contain readable information from the result.

Other result management systems group results into equivalence classes *without* associating a computed fingerprint with each result, and they denote each equivalence class with an arbitrary unique identifier. This identifier is opaque: it is ­*not* calculated from information stored in the result, and hence contains no readable information about the result.

Still other result management systems compute a fingerprint, associate an arbitrary unique identifier with the fingerprint, and use that identifier rather than the fingerprint to identify the equivalence class of results.

SARIF accommodates all these types of result management systems. Result management systems that compute fingerprints **SHOULD** populate the fingerprints property (§3.22.12). Result management systems that group results into equivalence classes based on an arbitrary unique identifier **SHOULD** populate the correlationGuid property (§3.22.4), regardless of whether they also compute a fingerprint.

### instanceGuid property

A result object **MAY** contain a property named instanceGuid whose value is a GUID-valued string (§3.5.3) defining a unique, stable identifier for the result.

Direct SARIF producers and SARIF converters **SHOULD NOT** set this property. A result management system **SHOULD** set this property when it ingests a SARIF log file. If it does so, then later, when a SARIF consumer retrieves results in SARIF format from the result management system, the result management system **SHALL** set this property to the value it assigned.

A result management system **MAY** store multiple results with identical fingerprints (see §3.22.12 and [Appendix B](#AppendixFingerprints)), but the instanceGuid properties for those results **SHALL** be distinct.

### correlationGuid property

A result object **MAY** contain a property named correlationGuid whose value is a GUID-valued string (§3.5.3) that is shared by all results that are considered logically identical, and that is different between any two results that are considered logically distinct.

Direct SARIF producers and SARIF converters **SHOULD NOT** set this property. A result management system **MAY** set this property when it ingests a SARIF log file. If it does so, then later, when a SARIF consumer retrieves results in SARIF format from the result management system, the result management system **MAY** set this property to the value it assigned.

NOTE: correlationGuid and fingerprints (§3.22.12) provide two different ways for result management systems to associate results that are logically identical. See §3.22.2 for more information.

### ruleId property

fDepending on the circumstances, a result object either **SHALL** or **SHALL NOT** contain a property named ruleId whose value is a hierarchical string (§3.5.4) consisting of one or more components, the first of which is the stable, opaque identifier of the rule that was evaluated to produce the result.

NOTE 1: ruleId will usually consist entirely of the rule’s stable opaque identifier. In some cases, it might be helpful to specify additional hierarchical components to more precisely describe the rule violation.

A SARIF viewer or result management system **MAY** use the additional hierarchical components to allow a user to suppress a subset of the violations of a given rule. A result result management system **MAY** also use the additional components to more precisely match results between runs.

EXAMPLE: In this example, the first result describes a violation of rule CA2101. Its ruleId consists entirely of the rule’s identifier. The second and third results both describe violations of rule CA5350. Each of their ruleIds specifies an additional hierarchical component that more precisely describes the rule violation. Note that the ruleIndex (§3.22.6) for both those results is 1; despite the additional hierarchical components in ruleId, both of those results describe violations of the same rule.

A SARIF viewer or result management system might allow a user to suppress, for example, only those violations of rule CA5350 which specify sha-1 as the second hierarchical component of ruleId; that is, to allow the use of SHA-1 but still warn about the uses of other weak cryptographic algorithms.

{

"results": [

{

"ruleId": "CA2101",

"ruleIndex": 0

},

{

"ruleId": "CA5350/md5",

"ruleIndex": 1

},

{

"ruleId": "CA5350/sha-1",

"ruleIndex": 1

}

],

"resources": {

"rules": [

{

"id": "CA201",

"shortDescription": {

"text": "Specify marshaling for P/Invoke string arguments"

}

},

{

"id": "CA5350",

"shortDescription": {

"text": "Do not use weak cryptographic algorithms"

}

}

]

}

]

Direct producers **SHALL** emit ruleId.

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

NOTE 2: If a converter were to synthesize a rule id, then two converters for the same tool might synthesize them differently, leading to loss of interoperability. For example, a SARIF consumer might mistakenly conclude that two issues, found in SARIF log files created by those two converters, were logically distinct because they had different rule ids.

### ruleIndex property

Depending on the circumstances, a result object either **SHOULD** or **SHALL NOT** contain a property named ruleIndex whose value is a non-negative integer that specifies the index of the relevant rule object (§3.40) in the resources.rules property (§3.13.15, §3.39.3) of the containing run object (§3.13), or in an external resource file (§3.11.6.4) or external property file (§3.13.2.1). If this property is absent, it **SHALL** default to -1 (which is otherwise not a valid value for this property).

NOTE: If ruleIndex is absent, a SARIF consumer will not be able to locate metadata for the rule.

If the relevant rule object is present in resources.rules or in an external resource file or external property file, then ruleIndex **SHOULD** be present. Otherwise, ruleIndex **SHALL NOT** be present.

EXAMPLE 1: In this example, there is more than one rule with id CA1711. ruleIndex uniquely specifies the relevant rule, whether or not there are multiple rules with the same id.

{ # A run object (§3.13).

"results": [ # See §3.13.14.

{ # A result object (§3.22).

"ruleId": "CA1711", # See §3.40.3.

"ruleIndex": 1,

...

}

],

"resources": { # See §3.13.15.

"rules": [ # See §3.39.3.

{ # A rule object (§3.40).

"id": "CA1711", # See §3.40.3.

...

},

{ # Another rule object with the same rule id.

"id": "CA1711", # ruleIndex points to this one.

...

}

]

}

}

If ruleIndex and ruleId are both present, then the value of the id property (§3.40.3) of the rule object specified by ruleIndex either **SHALL** equal the value of ruleId , or it **SHALL** equal a substring of ruleId that starts at the beginning of the string and ends before the last forward slash.

EXAMPLE 2: In this example, the first result object is valid because its ruleId property equals the id property of the rule object specified by ruleIndex (which we will refer to as theRule). The second result object is also valid because theRule.id equals all but the last component of ruleId. The third result object is invalid because theRule.id is not a “component-wise” prefix of ruleId.

{ # A run object (§3.13).

"results": [ # See §3.13.14.

{ # A result object (§3.22).

"ruleId": "abc/def", # See §3.40.3.

"ruleIndex": 0,

},

{

"ruleId": "abc/def/ghi",

"ruleIndex": 0

},

{

"ruleId": "abc/defg", # INVALID: Not a "component-wise" prefix of

"ruleIndex": 0 # theRule.id.

}

],

"resources": { # See §3.13.15.

"rules": [ # See §3.39.3.

{ # A rule object (§3.40).

"id": "abc/def", # See §3.40.3.

...

},

...

]

}

}

### kind property

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

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

* "pass": The rule specified by the ruleId property (§3.22.3) was evaluated, and no 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.

NOTE 1: This value is used by proof-based tools.Sometimes such a tool can prove that there is no violation (kind = "pass"), sometimes it can prove that there is a violation (kind = "fail"), and sometimes it does not detect a violation but is unable to prove that there is none (kind = "open"). If kind is "open", the user should add additional assertions to enabe the tool to determine if there is a violation.

* "notApplicable": The rule specified by the ruleId property was not evaluated, because it does not apply to the analysis target.

EXAMPLE: 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",

"kind": "notApplicable",

"message": {

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

because it is not a 32-bit binary."

},

"locations": [

{

"physicalLocation": {

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

}

}

]

}

]

* "review": The result requires review by a human user to decide if it represents a problem.

NOTE 2: This value is used by tools that are unable to check for certain conditions, but that wish to bring to the user’s attention the possibility that there might be a problem. For example, an accessibility checker might produce a result with the message "Do not use color alone to highlight important information," with kind = "review". A user might address this issue by visually inspecting the UI.

* "fail": The result represents a problem whose severity is specified by the level property (§3.22.8).

If kind is absent, it **SHALL** default to "fail".

If level has any value other than "none" and kind is present, then kind **SHALL** have the value "fail".

### 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:

* "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.
* "note": A purely informational log entry.
* "none": The concept of “severity” does not apply to this result because the kind property (§3.22.7) has a value other than "fail".

If the value of level is "note", then if the note relates to a particular rule, the ruleId property (§3.22.5) **SHOULD** be present; otherwise ruleId **SHALL** be absent.

EXAMPLE 1: In this example, the tool reports an observation about the code that does not represent a problem.

"results": [

{

"ruleId": "ABC0002",

"kind": "fail",

"level": "note",

"message": {

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

the parameter name 'start'."

},

"locations": [

{

"physicalLocation": {

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

"region": {

"startLine": 6

}

}

}

]

}

]

EXAMPLE 2: In this example, the tool reports information that is relevant to a particular rule but does not represent an observation about the code. Also in this example, kind is absent, and since level is present, kind defaults to "fail".

"results": [

{

"ruleId": "ABC0003",

"level": "note",

"message": {

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

}

}

]

EXAMPLE 3: 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 kind property has the value "fail" and level is absent, then level **SHALL** default to the defaultLevel property (§3.41.3) of the ruleConfiguration object (§3.41) contained in the configuration property (§3.40.12) of the rule object (§3.40) specified by this result object's ruleId property (§3.22.3).

In that case, if the run object (§3.13) containing this result does not include a resources.rules property (§3.13.15, §3.39.3) (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 level property **SHALL** default to "warning".

If kind has any value other than "fail", then if level is absent, it **SHALL** default to "none", and if it is present, it **SHALL** have the value "none".

### message property

A result object **SHALL** contain a property named message whose value is a message object (§3.11) that describes the result.

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 1: 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."

}

}

]

If the message object contains a messageId property (§3.11.9), the value of messageId **SHALL** identify the message within the rule metadata for the rule used in this result. If resources.rules (§3.13.15, §3.39.3) is present on the containing run object (§3.13), then message.messageId **SHALL** equal one of the property names in the messageStrings property (§3.40.8) of the rule object (§3.40) whose property name within resources.rules equals the ruleId property (§3.22.5) of this result object. message.messageId **MAY** also equal one of the property names in the richMessageStrings property (§3.40.9) of that rule object.

NOTE: This interpretation of message.messageId differs from its interpretation elsewhere in this specification. In every other message object in this specification, messageId refers to a string within the resources.messageStrings property of the current run object (see §3.11.9, and see §3.11.6.2 and §3.11.6.4 for details of the resource string lookup procedure). It is only result.message.messageId that refers to a string within resources.rules[*ruleId*].messageStrings.

EXAMPLE 2: In this example, message.messageId refers to the property named default defined in both the messageStrings property and the richMessageStrings property of the rule object identified by "CA2101".

{ # A run object (§3.13).

"results": [

{ # A result object (§3.22).

"ruleId": "CA2101",

"message": {

"messageId": "default"

},

...

}

],

"resources": { # A resources object (§3.39).

"rules": [

{

"id": "CA2101",

"messageStrings": {

"default": "The default message for this rule.",

"special": "Another message for this rule, for special cases."

},

"richMessageStrings": {

"default": "The default message for \*\*this\*\* rule.",

"special": "Another message for this rule, for \*\*special\*\* cases."

}

}

]

}

}

### locations property

A result object **SHOULD** contain a property named locations whose value is an array of zero or more location objects (§3.23), each of which specifies a location where the result occurred.

If locations is absent, it **SHALL** default to an empty array.

NOTE 1: In rare circumstances, it might not be possible to specify a location for a result. However, the locations property contains 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.

NOTE 2: The locations array is not defined to contain unique (§3.7.2) 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 location objects.

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 might contain a single result object, which would 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.  
  
EXAMPLE 4: 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.

### analysisTarget property

If the analysis target differs from the result file, a result object **SHOULD** contain a property named analysisTarget whose value is a fileLocation object (§3.4) that specifies the analysis target.

If the analysis target and the result file are the same, the analysisTarget property **SHOULD** be absent.

EXAMPLE: In this example, the tool’s analysis target was the file mouse.c. In the course of the scan, the tool detected a result in the included file mouse.h.

{ # A result object (§3.22).

"analysisTarget": { # A fileLocation object (§3.4

"uri": "input/mouse.c",

"uriBaseId": "SRCROOT"

},

"locations": [ # See §3.22.10.

{ # A location object (§3.23).

"physicalLocation": { # See §3.23.3.

"fileLocation": { # A fileLocation object.

"uri": "input/mouse.h",

"uriBaseId": "SRCROOT"

},

"region": {

"startLine": 42

}

}

}

]

}

### fingerprints property

A result object **MAY** contain a property named fingerprints whose value is an object (§3.6).

Each property value in this object **SHALL** be a string that provides a stable identifier for the result. This identifier **SHALL**, to the extent that it is feasible, be the same for all results that are logically identical, and different for any two results that are logically distinct. This requirement is intended to ensure that a fingerprint is resistant to changes that do not affect the logical identity of the result, such as the location of the root of a source code enlistment, or the line number where a result appears in a source file.

Each property name in this object **SHALL** be a versioned hierarchical string (§3.5.4.2). A result management system **MAY** use the property names to identify the method used to calculate the fingerprint.

EXAMPLE: In this example, the producer has calculated a fingerprint using version 2 of a fingerprinting method it refers to as "contextRegionHash":

{

"fingerprints": {

"contextRegionHash/v2": "097886bc876fe"

}

}

When a result management system uses fingerprint information to determine whether two results are logically identical, it **SHOULD** use the latest version of the fingerprint available in both results.

EXAMPLE 2: In this example, one result has values for versions 1 and 2 of the “context region hash” fingerprint. Another result has values for versions 2 and 3. A result management system would use version 2 (the greatest common version) to compare the two results.

{ # A run object (§3.13).

"results": [ # See §3.13.14.

{ # A result object (§3.22).

"fingerprints": {

"contextRegionHash/v1": "1234567900abc"

"contextRegionHash/v2": "234567900abcd"

},

{

"fingerprints": {

"contextRegionHash/v2": "234567900abcd"

"contextRegionHash/v3": "34567900abcde"

}

]

}

This property is an array, rather than a single string, to allow a result management system to select among a variety of methods for deciding whether two results are logically identical or logically distinct.

A direct SARIF producer **SHOULD NOT** populate this property. A SARIF converter **MAY** populate this property if the analysis tool’s native output format provides a value that qualifies as a fingerprint (a stable identifier for the result). A result management system **MAY** populate this property when it ingests a SARIF file. If it does so, then later, when a SARIF consumer retrieves results in SARIF format from the result management system, the result management system **MAY** set this property to the value it assigned.

[Appendix B](#AppendixFingerprints) provides requirements for how a result management system computes fingerprints.

NOTE: fingerprints and correlationGuid (§3.22.4) provide two different ways for result management systems to associate results that are logically identical. See §3.22.2 for more information.

### partialFingerprints property

A result object **MAY** contain a property named partialFingerprints whose value is an object (§3.6).

Each property value in this object **SHALL** be a string that contributes to the stable, unique identity, or “fingerprint,” of the result (see §3.22.12). [Appendix B](#AppendixFingerprints) explains how a result management system can compute these fingerprints.

Each property name in this object **SHALL** be a versioned hierarchical string (§3.5.4.2). A SARIF producer **MAY** use the property name to identify the nature of the information used to compute the partial fingerprint.

EXAMPLE 1: In this example, the producer has calculated a partial fingerprint using version 3 of a partial fingerprint value it refers to as "prohibitedWordHash":

{ # A result object (§3.22).

"partialFingerprints": {

"prohibitedWordHash/v3": "097886bc876fe"

}

}

When a result management system uses partial fingerprint information to determine whether two results are logically identical, it **SHOULD** use the latest version of the partial fingerprint available in both results.

EXAMPLE 2: In this example, one result has values for versions 1 and 2 of the “prohibited word hash” partial fingerprint. Another result has values for versions 2 and 3. A result management system would use version 2 (the greatest common version) to compare the two results.

{ # A run object (§3.13).

"results": [ # See §3.13.14.

{ # A result object (§3.22).

"partialFingerprints": {

"prohibitedWordHash/v1": "1234567900abc"

"prohibitedWordHash/v2": "234567900abcd"

},

{

"partialFingerprints": {

"prohibitedWordHash/v2": "234567900abcd"

"prohibitedWordHash/v3": "34567900abcde"

}

]

}

To make use of the information, if any, embodied in the property names, a result management system requires knowledge of the naming convention used by the SARIF producer. A result management system with that knowledge **MAY** use the property names to decide which partial fingerprints to include in its fingerprint computation. A result management system lacking that knowledge **SHALL** include all the partial fingerprints in its fingerprint computation.

Because result management systems 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 partial fingerprint **SHOULD** adhere to the following guidelines:

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

EXAMPLE 3: In this example, a SARIF-producing document checker has computed two partial fingerprints, one being a hash of a word that should not appear in a document, and the other being a hash of the document’s language.

{ # A result object

...

"partialFingerprints": {

"wordHash":

"2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae",

"langHash":

"5c49f88dafe66e0ecdca8f682ae0b38c38ccd3ad464e3358e899beca88c18560"

}

}

EXAMPLE 4. In this example, the SARIF producer has computed a single partial fingerprint. It has chosen an arbitrary value for the corresponding property name.

{ # A result object

...

"partialFingerprints": {

"1": "56eaf900cc8f6"

}

}

### codeFlows property

A result object **MAY** contain a property named codeFlows whose value is an array of zero or more codeFlow objects (§3.28). The codeFlows property is intended for use by analysis tools that provide execution path details that illustrate a possible problem in the code.

If codeFlows is absent, it **SHALL** default to an empty array.

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

### graphs property

A result object **MAY** contain a property named graphs whose value is an object (§3.6) each of whose property values is a graph object (§3.30), and each of whose property names equals the id property (§3.30.2) of that graph object. A graph object represents a directed graph: a network of nodes and directed edges that describes some aspect of the structure of the code (for example, a call graph).

A graph object defined at the result level **SHALL** be referenced only by graphTraversal objects (§3.33) defined in the graphTraversals property (§3.22.16) of the result object in which it is defined. This contrasts with graph objects defined at the run level (§3.13.13), which **MAY** be referenced by graphTraversal objects defined in the graphTraversals property of any result object in the containing run.

### graphTraversals property

If a result object contains a graphs property (§3.22.15), or if its containing run object (§3.13) contains a graphs property (§3.13.13), then the result object **MAY** contain a property named graphTraversals whose value is an array of one or more unique (§3.7.2) graphTraversal objects (§3.33). If neither the result object nor its containing run object contains a graphs property, the graphTraversals property **SHALL** be absent. A graph traversal is a path through the code that visits one or more nodes in a specified graph.

### stacks property

A result object **MAY** contain a property named stacks whose value is an array of stacks or more stack objects (§3.35). The stacks property is intended for use by analysis tools that collects call stack information in the process of producing results.

If stacks is absent, it **SHALL** default to an empty array.

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 zero or more location objects (§3.23), each of which represents a location relevant to understanding the result.

If relatedLocations is absent, it **SHALL** default to an empty array.

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": [

{

"physicalLocation": {

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

"region": {

"startLine": "6",

"startColumn": "10"

}

}

}

],

"relatedLocations": [ # An array of location objects

# (§3.23)

{ # A location object.

"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 either null or an array of zero or more unique (§3.7.2) strings, each of which specifies a reason the result should be “suppressed.”

If suppressionStates is absent, it **SHALL** default to null.

An empty array **SHALL** mean that the result should not be suppressed. null **SHALL** mean that it is not known whether the result should be suppressed.

The supported reasons for suppressing a result are:

* The developer has suppressed the result in the source code (see §3.22.19.2).
* The result is marked as suppressed in an external store such as a database (see §3.22.19.3).

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.

#### 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, which we refer to as the “baseline run.”

If baselineInstanceGuid (§3.13.5) is present on the containing run object (§3.13), its value **SHALL** specify the baseline run.

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 run.
* "unchanged": This result was detected both in the current run and in the baseline run, and it did not change between those two runs in any way that the tool considers significant.
* "updated": This result was detected both in the current run and in the baseline run, but it changed between those two runs in a way that the tool considers significant.
* "absent": This result was detected in the baseline run but was not detected in the current run.

NOTE 1: The purpose of baselineState 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 logically “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. See also the description of the fingerprints (§3.22.12) and partialFingerprints (§3.22.13) properties.

An analysis tool that works together with such a result management system can use the fingerprint to determine whether two results are logically the same; two results with the same fingerprint are considered logically the same.

NOTE 2: A result management system might respond to a “new” result by filing an issue in a bug tracking system. It might respond to an “updated” result by editing the details of an existing issue in the bug tracking system, or by attaching an updated SARIF log to the issue. It might respond to an “absent” result by resolving the issue. It might take no action at all for an “unchanged” issue, or it might simply update its internal information about the range of runs that contained the result.

If baselineState is present on any result object in the containing run, it **SHALL** be present on every such result object.

NOTE 3: The presence of baselineState on any result implies that the SARIF producer performed a comprehensive comparison between the results in the current run and those in some previous run. A SARIF consumer is entitled to expect that the differencing operation produced a baselineState value for every result.

This is conceptually similar to a tool that compares two text files, and for every line, concludes that it exists in the left-hand file, the right-hand file, or both. The tool must provide this information for every line in both files; it cannot leave some lines “undetermined.”

### rank property

A result object **MAY** contain a property named rank whose value is a number between 0.0 and 100.0 inclusive, representing the priority or importance of the result. 0.0 is the lowest priority and 100.0 is the highest.

If rank is absent, the ruleIndex property (§3.22.6) of the containing result object is present,

and the rule object designated by ruleIndex contains configuration.defaultRank (§3.40.12, §3.41.4), then rank **SHALL** default to the value of configuration.defaultRank.

Otherwise, rank **SHALL** default to -1.0.

NOTE: rank values produced by different tools are in general not commensurable. If Tool A produces one result with rank 0.65 and a second result with rank 0.70, the consumer is entitled to assume that the second result is of higher priority than the first. But if Tool A produces a result with rank 0.65 and Tool B produces a result with rank 0.70, the result produced by Tool B might or might not be of higher priority than the result produced by Tool A. In an engineering system that aggregates results from multiple tools, rank values might need to be adjusted, either automatically or by end users, so that rank values from different tools can be interleaved in a meaningful way.

### attachments property

A result object **MAY** contain a property named attachments whose value is an array of zero or more unique (§3.7.2) attachment objects (§3.18), each of which describes a file relevant to the detection of the result. The array elements **SHALL** refer to distinct files.

NOTE: For an illustration of the requirement for distinct files, see the examples in §3.17.5.

If attachments is absent, it **SHALL** default to an empty array.

### workItemUris property

A result object **MAY** contain a property named workItemUris whose value is either null or an array of zero or more unique (§3.7.2) strings, each containing the absolute URI [[RFC3986](#RFC3986)] of a work item associated with this result.

If workItemUris is absent, it **SHALL** default to null.

An empty array **SHALL** mean that there are no work items associated with this result. null **SHALL** mean that the set of work items associated with this result, if any, is not known.

NOTE: Result management systems are likely to generate work items from at least some of the results in a SARIF log file. Depending on the engineering system, these work items might take the form of Git issues, Jira tickets, TFS work items, or the equivalent in other work item tracking systems.

### hostedViewerUri property

A result object **MAY** contain a property named hostedViewerUri whose value is a string containing an absolute URI [[RFC3986](#RFC3986)] at which the result can be viewed. The URI **SHALL** be valid as of the time the tool generated this result. It is not guaranteed to be valid at later times (for example, the hosting environment might not keep results older than a specified age).

NOTE: This property can be used by tools that provide an online viewing experience for the results they generate. This experience might be specifically designed to display the results from that tool, as opposed to a generic SARIF viewer that displays results from any tool that produces SARIF.

### provenance property

A result object **MAY** contain a property named provenance whose value is a resultProvenance object (§3.38) that contains information about how and when the result was detected.

### fixes property

A result object **MAY** contain a property named fixes whose value is an array of one or more unique (§3.7.2) fix objects (§3.41).

If fixes is absent, it **SHALL** default to an empty array.

### occurrenceCount property

A result object **MAY** contain a property named occurrenceCount whose value is a positive integer specifying the number of times a result with this result object’s correlationGuid (§3.22.4) has been observed.

NOTE: This property is intended for the scenario where multiple SARIF files are being merged into a single SARIF file, with the intent that each logically distinct result (§3.22.2) occurs only once in the merged file. In that case, the system performing the merge would select one occurrence of each logically distinct result to serve as the exemplar for that class of results, and it would set occurrenceCount on that instance to the number of times a result with that correlationGuid occurred in the input files.

## location object

### General

A location object describes a location. Depending on the circumstances, a location object is described by physical location (§3.24), a logical location (§3.23.4), both, or in rare circumstances, neither (see below).

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 (§3.23.4) is particularly convenient for fingerprinting.

In rare circumstances, there might be neither physical nor logical location information available for a location object. See §3.37.2 for an example. In that case, the location object **SHOULD** contain a message property (§3.23.6) explaining the significance of this “location.”

### Constraints

If a location object specifies logical location information, at least one of the fullyQualifiedLogicalName property (§3.23.4) or the logicalLocationIndex property (§3.23.5) **SHALL** be present; they **MAY** both be present.

NOTE: Providing both fullyQualifiedName and logicalLocationIndex makes the log file more readable at the expense of increased size. Providing only logicalLocationIndex reduces log file size but makes it less readable to an end user, who has to determine the fully qualified name by locating the logicalLocation object (§3.27) at the index within run.logicalLocations (§3.13, §3.13.12) specified by logicalLocationIndex.

### physicalLocation property

If physical location information is available, a location object **SHALL** contain a property named physicalLocation whose value is a physicalLocation object (§3.24) that identifies the file within which the location lies. If physical location information is not available, physicalLocation **SHALL** be absent.

### fullyQualifiedLogicalName property

Depending on the circumstances, a location object either **SHALL**, **SHALL NOT**, or **MAY** contain a property named fullyQualifiedLogicalName whose value is a string containing the fully qualified name of the logical location, if any, described by this location object.

If this location object does not describe a logical location, then fullyQualifiedLogicalName **SHALL NOT** be present.

Throughout the remainder of this section, the location object is assumed to describe a logical location.

If the logicalLocationIndex property (§3.23.5) is absent, then fullyQualifiedLogicalName **SHALL** be present.

NOTE: This ensures that there is a way to locate information about the logical location specified by the location object.

If logicalLocationIndex is present, then fullyQualifiedLogicalName **MAY** be present.

The format of fullyQualifiedLogicalName **SHALL** follow the naming rules for fully qualified logical locations described in §3.27.2.

EXAMPLE 1: C: create\_process

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

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

If the array-valued logicalLocations property (§3.13.12) of the containing run object (§3.13) is present, fullyQualifiedLogicalName **SHOULD** equal the fullyQualifiedNameProperty (§3.27.4) of one of the logicalLocations objects (§3.27) in the array.

It is possible for two or more distinct logical locations to have the same fully qualified logical name.

NOTE: This is an extremely rare corner case.

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. However, they would be distinguished by their parentIndex properties:

"logicalLocations": [

{

"name": "B",

"fullyQualifiedName": "A::B",

"kind": "namespace",

"parentIndex": 1

},

{

"name": "A",

"kind": "namespace"

},

{

"name": "B",

"fullyQualifiedName": "A::B",

"kind": "type",

"parentIndex": 3

},

{

"name": "A",

"kind": "namespace"

}

]

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:

* run.logicalLocations might not be present.
* It allows a SARIF viewer to display the logical location in a way that is easily understood by users.
* As mentioned in §3.23.1, 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”).

### logicalLocationIndex property

Depending on the circumstances, a location object either **SHOULD** or **SHALL NOT** contain a property named logicalLocationIndex whose value is a non-negative integer that specifies the index within the array-valued logicalLocations property (§3.13.12) of the containing run object (§3.13) of the logicalLocation object (§3.27) that describes the logical location, if any, specified by this location object. If logicalLocationIndex is absent, it **SHALL** default to -1 (which is otherwise not a valid value for this property).

NOTE: If logicalLocationIndex is absent, the SARIF consumer will not be able to locate information about the logical location.

If this location object does not describe a logical location, then logicalLocationIndex **SHALL NOT** be present.

Throughout the remainder of this section, the location object is assumed to describe a logical location.

If run.logicalLocations is absent, or if it does not contain an entry for the logical location described by this location object, then logicalLocationIndex **SHALL NOT** be present.

Otherwise, logicalLocationIndex **SHOULD** be present.

If logicalLocationIndex and fullyQualifiedLogicalName (§3.23.4) are both present, then the value of the fullyQualifiedName property (§3.27.4) of the logicalLocation object specified by logicalLocationIndex **SHALL** equal the value of fullyQualifiedLogicalName.

### message property

A location object **MAY** contain a property named message whose value is a message object (§3.11) relevant to the location.

### annotations property

A location object **MAY** contain a property named annotations whose value is an array of zero or more unique (§3.7.2) region objects (§3.25), each of which describes a region within the file specified by the location object that is relevant to the location. Each of these region objects **SHOULD** contain a message property (§3.25.14) that explains the relevance of the region to the location.

If annotations is absent, it **SHALL** default to an empty array.

EXAMPLE: Consider a location object which describes the declaration statement

int x = (y + z) \* q;

If the analysis tool wanted to emphasize the expression (y + z), it might set the annotations property to:

"annotations": [ # An array of region objects.

{ # A region object (§3.25).

"startLine": 12,

"startColumn": 13,

"endColumn": 19,

"message": {

"text": "(y + z) = 42"

}

}

]

## 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 containing result object (§3.22). 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].threadFlows[0].locations[0].physicalLocation

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

The purpose of the id property is to enable an embedded link (§3.11.4) within a message object (§3.11) to refer to the location. If no message object within the containing result object refers to this location *via* an embedded link, the id property does not need to appear.

### fileLocation property

A physicalLocation object **SHALL** contain a property named fileLocation whose value is a fileLocation object (§3.3) that represents the location of the file.

### region property

A physicalLocation object **MAY** contain a property named region whose value is a region object (§3.25) that represents a relevant portion of the file. In particular, if the physicalLocation object occurs within the locations property (§3.22.10) of a result object (§3.22), the region property **SHALL** specify the region within the file where the result was detected.

EXAMPLE 1: In this example, a physicalLocation object specifies the location where a result was detected. Its region property specifies the portion of the file where the result was detected.

{ # A result object (§3.22).

"locations": [ # See §3.22.10.

{ # A location object (§3.23).

"physicalLocation": { # See §3.23.3.

"fileLocation": { # A physicalLocation object.

"uri": "ui/window.c",

"uriBaseId": "SRCROOT"

},

"region": { # The region specifies the portion of the file

"startLine": 42 # where the result was detected.

}

}

}

]

}

If the physicalLocation object specifies a location in a nested file, then the region property **SHALL** specify the location with respect to the innermost nested file.

EXAMPLE 2: 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.

### contextRegion property

If a physicalLocation object contains a region property (§3.24.4), it **MAY** also contain a property named contextRegion whose value is a region object (§3.25) which specifies a region that is a proper superset of the region specified by the region property. If region is absent, contextRegion **SHALL** be absent.

The purpose of contextRegion is to enable a viewer to provide visual context when displaying a portion of a file.

EXAMPLE In this example, an analysis tool detected a result on line 42. The tool provides additional context SARIF viewers by specifying a range of content surrounding the result line.

{ # A result object (§3.22).

"locations": [ # See §3.22.10.

{ # A location object (§3.23).

"physicalLocation": { # See §3.23.3.

"fileLocation": { # A physicalLocation object.

"uri": "ui/window.c",

"uriBaseId": "SRCROOT"

},

"region": { # See §3.24.4.

"startLine": 42,

"snippet": {

"text": "int n = m + 1;"

}

},

"contextRegion": {

"startLine": 41,

"endLine": 43,

"snippet": {

"text": "int m;\nint n = m + 1\n\n"

}

}

}

}

]

}

## region object

### General

A region object represents a region, that is, a contiguous portion of a file.

The region object defines both “text properties” and “binary properties.” The text properties represent a region as a contiguous range of zero or more characters (a “text region”). The binary properties represent a region as a contiguous range of zero or more bytes (a “binary region”).

For regions in text files, a region object **SHOULD** contain text properties and **MAY** also contain binary properties. If both text properties and binary properties are present, they **SHALL** specify the identical range of bytes in the file, as determined by the file’s character encoding.

For regions in binary files, a region object **SHALL** contain binary properties and **SHALL NOT** contain text properties.

If any text properties are present, enough text properties **SHALL** be present to fully specify a text region (see §3.25.2). If any binary properties are present, then enough binary properties **SHALL** be present to fully specify a binary region (see §3.25.3).

### Text regions

NOTE 1: The examples in this section assume a text file with the following contents:

abcd\r\nefg\r\nhijk\r\nlmn\r\n

Breaking the lines for the sake of readability, the contents are:

abcd\r\n

efg\r\n

hijk\r\n

lmn\r\n

The file contains four lines, each of which ends with the two-character newline sequence "\r\n", which is explicitly displayed for clarity.

The line number of the first line in a text file **SHALL** be 1. The column number of the first character in each line **SHALL** be 1. The character offset of the first character in the file **SHALL** be 0.

The values of text properties **SHALL NOT** depend on the presence or absence of a byte order mark (BOM) at the start of the file.

Column numbers are expressed in the measurement unit specified by the columnKind property (§3.13.19) of the containing run object (§3.13).

A SARIF viewer **MAY** choose to present column numbers that match the visual offset of each character from the beginning of the line. These “visual” column numbers might not match the column numbers contained in the SARIF file.

NOTE 2: Such a mismatch might occur if, for example, the line contains a tab character, or an accented character represented by a base character plus a combining character.

A text file’s character encoding determines the number of bytes that represent each character, and therefore determines the range of bytes represented by a text region. A SARIF consumer **SHALL** consider a file to have the encoding specified by file.encoding (§3.21.9), if present, or else by run.defaultFileEncoding (§3.13.16), if present. If neither is present, the consumer **MAY** use any heuristic or procedure to determine the encoding, including (for example) prompting the user.

NOTE 3: If a consumer incorrectly determines a file’s encoding, it might not display the file correctly. For example, when it attempts to highlight a region, it might highlight an incorrect range of characters.

A text region **MAY** be specified in two ways:

* By means of the “line/column” properties startLine (§3.25.5), startColumn (§3.25.6), endLine (§3.25.7), and endColumn (§3.25.8).
* By means of the “offset/length” properties charOffset (§3.25.9) and charLength (§3.25.10).

A text region **SHALL** specify both its start (the location of its first character) and its end (the location of its last character).

A text region does not include the character specified by endColumn (see §3.25.8).

EXAMPLE 1: The following regions (among others) all specify the range of characters "bc".

{

"startLine": 1,

"startColumn": 2,

"endLine": 1,

"endColumn": 4 # The region excludes the character at endColumn.

}

{

"charOffset": 1,

"charLength": 2

}

{

"startLine": 1,

"startColumn": 2,

"endLine": 1,

"endColumn": 4,

"charOffset": 1,

"charLength": 2

}

EXAMPLE 2: The following region is invalid, even though it might appear to specify the same range of characters "bc" as in EXAMPLE 1:

{

"startLine": 1,

"charOffset": 1, # Specifies the "b"

"endColumn": 4 # Specifies the column one past the "c"

}

This is because the line/column properties and the offset/length properties, taken independently, specify different regions:

* "startColumn" is absent, and so defaults 1 (see §3.25.6).
* "endLine" is absent, and so defaults to "startLine", which in this example is 1 (see §3.25.7).
* "charLength" is absent, and so defaults to 0 (see §3.25.10).

In summary, the above region is equivalent to the region

{

"startLine": 1,

"startColumn": 1,

"endLine": 1,

"endColumn": 4,

"charOffset": 1,

"charLength": 0

}

Now we can see that the line/column properties represent the range of characters "abc", while the offset/length properties represent an insertion point before the character "b" (see §3.25.10). Those two regions are not the same, and so the region is invalid.

Incidentally, *neither* of those regions is the same as the "bc" one might expect if it were valid to compose a region partially from a mixture of line/column properties and offset/length properties.

If a region spans more than one line, it **SHALL** include the newline sequences of all but the last line in the region.

EXAMPLE 3: The region

{ "startLine": 2, "endLine": 3 }

includes the characters "efg\r\nhijk".

A region of length 0 is referred to as an “insertion point.” An insertion point **MAY** be specified either by specifying charLength as 0, or by specifying the same values for startColumn and endColumn.

NOTE 4: This is consistent with the rule that a region does not include the character in column endColumn.

EXAMPLE 4: These regions (among others) specify an insertion point before the "b" on line 1.

{ "startLine": 1, "startColumn": 2, "endColumn": 2 }

{ "charOffset": 1, ""charLength": 0 }

EXAMPLE 5: These regions (among others) specify an insertion point at the beginning of the file:

{ "startLine": 1, "startColumn": 1, "endColumn": 1 }

{ "charOffset": 0, "charLength": 0 }

To specify an insertion point after the last character in a file, set endLine to the number of the last line in the file, and set endColumn to a value one greater than the number of characters on the line, *including* any trailing newline sequence..

EXAMPLE 6: These regions (among others) specify an insertion point at the very end of the file. Note that the last line contains the five characters (including the newline sequence) "lmn\r\n".

{ "startLine": 4, "startColumn": 6, "endColumn": 6 }

{ "charOffset": 22, "charLength": 0 }

### Binary regions

The byte offset of the first byte in a file **SHALL** be 0.

To specify a byte region, at least byteOffset (§3.25.11) **SHALL** be present. byteLength (§3.25.12) **MAY** also be present. byteOffset specifies the start of the region. byteLength specifies the end of the region. A byteLength value of 0 represents an insertion point before the byte specified by byteOffset.

### Independence of text and binary regions

The text-related and binary-related properties in a region object **SHALL** be treated independently. That is, the value of a text-related property **SHALL NOT** be inferred from the value of any set of binary-related properties, and *vice versa*.

EXAMPLE: This example is based on the sample text file show in NOTE 1 of §3.25.2. It represents invalid SARIF because the text-related and binary-related properties are inconsistent. At first glance they appear to be consistent because the byte at offset 2 is indeed on line 1:

{ "startLine": 1, "byteOffset": 2, "byteLength": 6 }

However, because the default values for the missing text-related properties are determined entirely from the existing text-related properties, and independently of any binary-related properties, this region is in fact equivalent to this one:

{

"startLine": 1,

"startColumn": 1, // Missing startColumn defaults to 1.

"endLine": 1, // Missing endLine defaults to startLine.

"endColumn": 5, // Missing endColumn defaults to (length of endLine + 1),

// exclusive of newline sequence.

"byteOffset": 2

"byteLength": 6

}

This makes it clear that the text-related and binary-related properties represent different ranges of bytes, and therefore the region is invalid.

### startLine property

When a region object represents a text region specified by line/column properties, it **SHALL** contain a property named startLine whose value is a positive integer equal to the line number of the line containing the first character in the region.

### startColumn property

When a region object represents a text region specified by line/column properties, it **MAY** contain a property named startColumn whose value is a positive integer equal to the column number of the first character in the region.

If startColumn is absent,it **SHALL** default to 1.

### endLine property

When a region object represents a text region specified by line/column properties, it **MAY** contain a property named endLine whose value is a positive integer equal to the line number of the line containing the last character in the region.

If endLine is absent, its value **SHALL** default to startLine.

### endColumn property

When a region object represents a text region specified by line/column properties, it **MAY** contain a property named endColumn whose value is an integer whose value is one greater than the column number of the last character in the region.

If endColumn is absent, it **SHALL** default to a value one greater than the column number of the last character on the line, excluding any newline sequence.

### charOffset property

When a region object represents a text region specified by offset/length properties, it **SHALL** contain a property named charOffset whose value is an integer equal to the zero-based character offset of the first character in the region from the beginning of the file.

### charLength property

When a region object represents a text region specified by offset/length properties, it **MAY** contain a property named charLength whose value is a non-negative integer equal to the number of characters in the region.

If charLength is absent, it **SHALL** default to 0, which **SHALL** be interpreted as an insertion point at the position specified by charOffset (§3.25.9)

The sum of charOffset and charLength **SHALL** be greater than or equal to 0 and less than or equal to the number of characters in the file.

A region whose charOffset is equal to the number of characters in the file and whose charLength is 0 is permitted and **SHALL** represent an insertion point at the end of the file.

### byteOffset property

When a region object represents a binary region, it **SHALL** contain a property named byteOffset whose value is an integer equal to the zero-based byte offset of the first byte in the region from the beginning of the file.

### byteLength property

When a region object represents a binary region, it **MAY** contain a property named byteLength whose value is an integer equal to the number of bytes in the region. If byteLength is absent, it **SHALL** default to 0, which **SHALL** be interpreted as an insertion point at the position specified by byteOffset (§3.25.11).

The sum of byteOffset and byteLength **SHALL** be greater than or equal to 0 and less than or equal to the number of bytes in the file.

A region object whose byteOffset equals the number of bytes in the file and whose byteLength is 0 is permitted, and **SHALL** represent an insertion point at the end of the file.

### snippet property

A region object **MAY** contain a property named snippet whose value is a fileContent object (§3.3) representing the portion of the file specified by the region object.

NOTE: The purpose of the snippet property is to allow a SARIF viewer to present the contents of the region even if the file from which it was taken is not available. It also allows an end user examining a SARIF log file to see the relevant file content without opening another file.

### message property

A region object **MAY** contain a property named message whose value is a message object (§3.11) containing a message relevant to the region.

A SARIF viewer **SHOULD** display this message when the user interacts with the region. For example, if the user hovers over the region with the mouse, the viewer might present the message as hover text.

### sourceLanguage property

If the region object represents a portion of a text file that contains source code, it **MAY** contain a property named sourceLanguage whose value is a hierarchical string (§3.5.4) that specifies the programming language in which this portion of the source code is written. If the region object does not represent a portion of a text file containing source code, then sourceLanguage **SHALL** be absent.

For the remainder of this section, we assume that the the region object represents a portion of a text file that contains source code.

NOTE: This property is intended to help SARIF viewers to render code snippets (§3.25.13) with appropriate syntax coloring. It is intended for use in mixed-language files, such as HTML files that contains JavaScript. For more information about this usage, see §3.21.10.

if sourceLanguage is absent, it **SHALL** default to the value of the sourceLanguage property (§3.21.10) of the file object (§3.21) which describes the file that contains the region. file.sourceLanguage in turn defaults to the defaultSourceLanguage property (§3.13.17) of the containing run object (§3.13). If all three of region.sourceLanguage, file.sourceLanguage, and run.defaultSourceLanguage are absent, the source language of the region object **SHALL** be taken to be unknown. In that case, a SARIF viewer **MAY** use any method or heuristic to determine the region’s source language, for example, by examining the file’s file name extension or MIME type, or by prompting the user.

For conventions and practices regarding the value of this property, see §3.21.10.2.

## rectangle object

### General

A rectangle object specifies a rectangular area within an image. When a SARIF viewer displays an image, it **SHOULD** indicate the presence of these areas, for example, by highlighting them or surrounding them with a border.

### top, left, bottom, and right properties

A rectangle object **SHALL** contain properties named top, left, bottom, and right, each of which contains a number (as defined by [[JSCHEMA01](#JSCHEMA01)]) specifying one of the coordinates of the rectangle within the image. These properties **SHALL** be measured in the image format’s natural units (for example, pixels for raster-based image formats). These values **MAY** be positive or negative, depending on the natural coordinate system of the image format. They **MAY** increase either from left to right or from right to left, and either from top to bottom or from bottom to top, again depending on the natural coordinate system of the image format.

NOTE: A number in JSON schema can take a variety of forms, including simple integers (42) and floating-point numbers (3.14).

### message property

A rectangle object **SHOULD** contain a property named message whose value is a message object (§3.11) containing a message relevant to this area of the image.

A SARIF viewer **SHOULD** display this message when the user interacts with the area. For example, if the user hovers over the area with the mouse, the viewer might present the message as hover text.

## logicalLocation object

### General

A logicalLocation object describes a logical location. A logical location is a location specified by a programmatic construct such as a namespace, a type, or a method, without regard to the physical location where the construct occurs.

logicalLocation objects occur as elements within the run.logicalLocations array (§3.13.12).

### Logical location naming rules

Every logical location has a “fully qualified logical name” (more briefly, a “fully qualified name”) that fully specifies the programmatic construct to which it refers. When programmatic constructs are nested (such as a method within a class within a namespace), the fully qualified name is typically a hierarchical identifier such as "N.C.F(void)" or "N::C::F(void)". We refer to the rightmost component of this hierarchical identifier as the “logical name” (more briefly, the “name”) of the logical location.

Logical location names and fully qualified names appear in various properties in the SARIF format:

* logicalLocation.name (§3.27.3): a logical name.
* logicalLocation.fullyQualifiedName (§3.27.4): a fully qualified logical name.
* location.fullyQualifiedLogicalName (§3.23.4): a fully qualified logical name.

Whenever possible, logical names and fully qualified logical names **SHALL** conform to the syntax of the programming language in which the programmatic construct specified by the logical location was expressed.

EXAMPLE 1: The fully qualified logical name of the C++ method f(void) in class C in namespace N is "N::C::f(void)". Its logical name is "f(void)".

This is not always possible, for two reasons:

* For certain values of logicalLocation.kind (§3.27.6), there is no language syntax to specify the fully qualified name.

EXAMPLE 2: Suppose the logical location is the local variable pBuffer in the C++ method "N::C::f(void)". logicalLocation.kind is "variable". There is no way to express the fully qualified name in C++. The SARIF producer might choose a fully qualified name such as "N::C::f(void)?pBuffer".

* For other values of logicalLocation.kind, it is sometimes but not always possible to express the logical location in language syntax.

EXAMPLE 3: Suppose the logical location is the anonymous callback function in this JavaScript function:

function click\_it() {

$("button").click(function(){

alert("Clicked!");

});

}

logicalLocation.kind is "function", for which it is sometimes possible to specify a fully qualified name. But there is no language syntax to express the name of an anonymous callback. The SARIF producer might choose a fully qualified name such as "click\_it?anon-1".

### name property

A logicalLocation object **SHALL** contain a property named name whose value is the logical name of the programmatic construct specified by this object. For example, this property might contain the name of a class or a method.

The name property **SHALL** be suitable for display and **SHALL** follow the naming rules for logical names described in §3.27.2.

NOTE: A C++ analysis tool might have available both the source code form of a function name and the compiler’s “decorated” function name (which encodes the function signature in a manner that is compiler-dependent and not easily readable). The tool would place the source code form of the function name in the name property, and the decorated name in the decoratedName property (§3.27.6).

EXAMPLE: In this C++ example, the fully qualified name is "b::c(float)", so "name" is the rightmost component, "c(float)".

{ # A logicalLocation object.

"name": "c(float)",

"fullyQualifiedName": "b:c(float)", # See §3.27.4.

"kind": "function" # See §3.27.6

}

### fullyQualifiedName property

Depending on the circumstances, a logicalLocation object either **SHALL** or **MAY** contain a property named fullyQualifiedName whose value is the fully qualified name of the logical location. This name **SHALL** follow the naming rules for fully qualified names described in §3.27.2.

If this logicalLocation object represents a top-level logical location, then fullyQualifiedName **MAY** be present. If present, it **SHALL** equal name; if absent, it **SHALL** default to name. If this object does not represent a top-level logical location, fullyQualifiedName **SHALL** be present.

### decoratedName property

A logicalLocation 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. Some compilers refer to this representation as a “mangled name.”

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

{ # A "logicalLocation" object

"name": "c(float)",

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

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

}

### 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"
* "resource"
* "type"
* "returnType"
* "parameter"
* "variable"

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

If a logical location is both a member and a type (for example, a nested class in C++ or C#), the value of kind, if present, **SHALL** be "type".

NOTE: The purpose of this property is to help result management systems group results that occur in the same logical location. If one result specifies the logical location “namespace A”, and another result specifies the logical location “class A”, the difference in the kind property between the two results tells the result management system to sort them into different groups.

### parentIndex property

If this logicalLocation object represents a nested logical location, then it **SHALL** contain a property named parentIndex whose value is a non-negative integer that specifies the index of the parent logicalLocation object within the run.logicalLocations array (§3.13.12).

If this logicalLocation object represents a top-level logical location, then parentKey **SHALL** be absent.

If this property is absent, it **SHALL** default to -1.

NOTE: parentIndex makes it possible to navigate from the logicalLocation object representing a nested logical location to the logicalLocation objects representing each of its parent logical locations in turn, up to the top-level logical location.

EXAMPLE: In this example, the logical location n::f(void) is nested within the top-level logical location n. The logicalLocation object representing n::f(void) contains a parentIndex property that points to the object representing n; the object representing n does not contain a parentIndex property.

{ # A run object (§3.13).

"logicalLocations": [ # See §3.13.12.

{

"name": "f(void)", # See §3.27.3.

"fullyQualifiedName": "n::f(void)", # See §3.27.4.

"kind": "function", # See §3.27.6.

"parentIndex": 1

},

{

"name": "n",

"kind": "namespace"

}

]

}

## codeFlow object

### General

A codeFlow object describes the progress of one more programs through one or more thread flows, which together result in the detection of a result in the system being analyzed. We define a thread flow as a temporally ordered sequence of code locations occurring within a single thread of execution, typically an operating system thread or a fiber. The thread flows in a code flow **MAY** lie within a single process, within multiple processes on the same machine, or within multiple processes on multiple machines.

EXAMPLE

{ # A result object (§3.22).

"codeFlows": [ # See §3.22.14.

{ # A codeFlow object (§3.28).

"message": { # See §3.28.2.

"text": "..."

},

"threadFlows": [ # See §3.28.3.

{ # A threadFlow object (§3.29).

"id": "thread-123", # See §3.29.2.

"message": { # See §3.29.3.

"text": "..."

},

"locations": [ # See §3.29.4.

{ # A threadFlowLocation object (§3.37).

"location": { # See §3.37.2.

"physicalLocation": { # See §3.23.3.

"fileLocation": {

"uri": "ui/window.c",

"uriBaseId": "SRCROOT"

},

"region": {

"startLine": 42

}

}

},

"state": { # See §3.37.6.

"x": "42",

"y": "54",

"x + y": "96"

},

"nestingLevel": 0, # See §3.37.7.

"executionOrder": 2 # See §3.37.8.

}

]

}

]

}

]

}

### message property

A codeFlow object **MAY** contain a property named message whose value is a message object (§3.11) relevant to the code flow.

### threadFlows property

A codeFlow object **SHALL** contain a property named threadFlows whose value is an array of one or more threadFlow objects (§3.29), each of which describes the progress of a program through a single thread of execution such as an operating system thread or a fiber.

## threadFlow object

### General

A thread flow is a sequence of code locations that specify a possible path through a single thread of execution such as an operating system thread or a fiber.

For an example, see §3.28.1.

### id property

A threadFlow object **MAY** contain a property named id whose value is a string that uniquely identifies this threadFlow within its containing codeFlow object (§3.28).

NOTE: A tool might choose to use an operating system thread id for this purpose. However, if thread ids are reused on a single machine, or if the code flow includes thread flows from more than one machine, the thread id might not be unique.

### message property

A threadFlow object **MAY** contain a property named message whose value is a message object (§3.11) relevant to the thread flow.

### locations property

A threadFlow object **SHALL** contain a property named locations whose value is an array of one or more threadFlowLocation objects (§3.37). 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 execution order that demonstrates the problem. 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.

## graph object

### General

A graph object represents a directed graph, a network of nodes and directed edges that describes some aspect of the structure of the code (for example, a call graph). graph objects **MAY** be defined both at the run level in run.graphs (§3.13.13) and at the result level in result.graphs (§3.22.15).

A path through a graph, called a “graph traversal,” is represented by a graphTraversal object (§3.33).

### id property

A graph object **MAY** contain a property named id whose value is a string that uniquely identifies the graph within its containing run.graphs property (§3.13.13) or result.graphs property (§3.22.15). The id property does not have to be unique across all graph objects in all result.graphs properties in the run.

If id is present, it **SHALL** equal the corresponding property name in the run.graphs or result.graphs object in which it appears. If id is absent, it **SHALL** default to that property name.

EXAMPLE: In this example, the first graph explicity states its id property ("g1"), which matches its property name. The second graphs’s id property defaults to the property name ("g2"). The third graph is invalid because its id property ("g4") does not match its property name ("g3").

{ # A run object (§3.13).

"graphs": {

"g1": { # id "g1" matches property name.

"id": "g1",

...

},

"g2": { #id is absent; defaults to property name "g2".

...

},

"g3": {

"id": "g4", # INVALID: id "g4" does not match property name "g3".

...

}

### description property

A graph object **MAY** contain a property named description whose value is a message object (§3.11) that describes the graph.

### nodes property

A graph object **MAY** contain a property named nodes whose value is an array of zero or more unique (§3.7.2) node objects (§3.31) which represent the nodes of the graph.

If nodes is absent, it **SHALL** default to an empty array.

### edges property

A graph object **MAY** contain a property named edges whose value is an array of zero or more unique (§3.7.2) edge objects (§3.32) which represent the edges of the graph.

If edges is absent, it **SHALL** default to an empty array.

## node object

### General

A node object represents a node in the graph represented by the containing graph object (§3.30).

### id property

A node object **SHALL** contain a property named id whose value is a string that uniquely identifies the node within the containing graph object (§3.30). id **SHALL** be unique among all nodes in the graph, regardless of nesting (see §3.31.5).

EXAMPLE: This graph is invalid because two nodes have the same id, even though the nodes are within unrelated nested graphs.

{ # A graph object (§3.30).

"nodes": [ # See §3.30.4.

{ # A node object.

"id": "n1",

"children": [ # See §3.31.5.

{

"id": "n3"

}

]

},

{

"id": "n2",

"children": [

{

"id": "n3" # INVALID: duplicate id.

}

]

}

],

...

}

### label property

A node object **MAY** contain a property named label whose value is a message object (§3.11) that provides a short description of the node.

### location property

A node object **SHOULD** have a property named location whose value is a location object (§3.23) that specifies the location associated with the node.

### children property

A node object **MAY** contain a property named children whose value is an array of zero or more unique (§3.7.2) node objects, referred to as “child nodes.”

If children is absent, it **SHALL** default to an empty array.

Child nodes are logically subordinate to their containing node, and form a “nested graph” within that node.

## edge object

### General

An edge object represents a directed edge in the graph represented by the containing graph object (§3.30).

### id property

An edge object **SHALL** contain a property named id whose value is a string that uniquely identifies the edge within the containing graph object (§3.30).

### label property

An edge object **MAY** contain a property named label whose value is a message object (§3.11) that provides a short description of the edge.

### sourceNodeId property

An edge object **SHALL** contain a property named sourceNodeId whose value is a string that identifies the source node (the node at which the edge starts). It **SHALL** equal the id property (§3.31.2) of one of the node objects (§3.31) in the containing graph object (§3.30). It **MAY** equal the id of any node within the graph, regardless of nesting (see §3.31.5).

EXAMPLE: In this example, an edge connects two nodes defined in unrelated nested graphs.

{ # A graph object (§3.30).

"nodes": [ # See §3.30.4.

{ # A node object.

"id": "n1",

"children": [ # See §3.31.5.

{

"id": "n3"

}

]

},

{

"id": "n2",

"children": [

{

"id": "n4"

}

]

}

],

"edges": [ # See §3.30.5.

{

"sourceNodeId": "n3", # Source node and target node are in separate

"targetNodeId": "n4" # nested graphs: ok.

}

],

...

}

### targetNodeId property

An edge object **SHALL** contain a property named targetNodeId whose value is a string that identifies the target node (the node at which the edge ends). It **SHALL** equal the id property (§3.31.2) of one of the node objects (§3.31) in the containing graph object (§3.30). It **MAY** equal sourceNodeId (§3.32.4).

## graphTraversal object

### General

A graphTraversal object represents a “graph traversal,” that is, a path through a graph specified by a sequence of connected “edge traversals,” each of which is represented by an edgeTraversal object (§3.34). For an example, see §3.33.5.

### graphId property

A graphTraversal object **SHALL** contain a property named graphId whose value is a string that equals the id property (§3.30.2) of the graph object (§3.30) being traversed.

The value of graphId **SHALL** equal the id of a graph object that occurs in the graphs property (§3.22.15) of the containing result object (§3.22), or the id of a graph object that occurs in the graphs property (§3.13.13) of the containing run object (§3.13), or both (in which case the graph object in result.graphs takes precedence).

### description property

A graphTraversal object **MAY** contain a property named description whose value is a message object (§3.11) that describes the graph traversal.

### initialState property

A graphTraversal object **MAY** contain a property named initialState whose value is an object (§3.6) each of whose properties represents the value of a relevant expression at the point of entry to the graph. This property, together with edgeTraversal.finalState (§3.34.4), enables a SARIF viewer to present a debugger-like “watch window” experience as the user traverses a graph.

For details of how properties within a “state” object are represented, see §3.37.6.

### edgeTraversals property

A graphTraversal object **MAY** contain a property named edgeTraversals whose value is an array of zero or more edgeTraversal objects (§3.34) which together represent the sequence of edges traversed during this graph traversal.

If edgeTraversals is absent, it **SHALL** default to an empty array.

The edgeTraversal objects **SHALL** be connected end to end; that is, the target node of every traversed edge **SHALL** equal the source node of the next edge.

EXAMPLE: In this example, the graphTraversal contains two edgeTraversal objects. The id of the first traversed edge is "e1", which connects node "n1" to node "n2". The id of the second traversed edge is "e3", which connects node "n2" to node "n4". This is a valid graph traversal because the target node of each traversed edge is the source node of the next.

This example also demonstrates the usage of graphTraversal.initialState (§3.33.4) and edgeTraversal.finalState (§3.34.4).

{ # A result object (§3.22).

"graphs": { # See §3.22.15.

"g1": { # A graph object (§3.30).

"id": "g1", # See §3.30.2.

"nodes": [ # See §3.30.4.

{ "id": "n1" }, # A node object (§3.31).

{ "id": "n2" },

{ "id": "n3" },

{ "id": "n4" }

],

"edges": [ # See §3.30.5.

{ # An edge object (§3.32).

"id": "e1", # See §3.32.2.

"sourceNodeId": "n1", # See §3.32.4.

"targetNodeId": "n2" # See §3.32.5.

},

{

"id": "e2",

"sourceNodeId": "n2",

"targetNodeId": "n3"

},

{

"id": "e3",

"sourceNodeId": "n2",

"targetNodeId": "n4"

}

]

}

},

"graphTraversals": [ # See §3.22.16.

{ # A graphTraversal object (§3.33).

"graphId": "g1", # See §3.33.2.

"initialState": { # See §3.33.4.

"x": "1",

"y": "2",

"x + y": "3”

},

"edgeTraversals": [ # See §3.33.5.

{ # An edgeTraversal object (§3.34).

"edgeId": "e1", # See §3.34.2.

"finalState": { # See §3.34.4.

"x": "4",

"y": "2",

"x + y": "6”

}

},

{

"edgeId": "e3",

"finalState": {

"x": "4",

"y": "7",

"x + y": "11”

}

}

]

}

]

}

## edgeTraversal object

### General

An edgeTraversal object represents the traversal of a single edge during a graph traversal.

### edgeId property

An edgeTraversal object **SHALL** contain a property named edgeId whose value is a string which equals the id property (§3.32.2) of one of the edge objects (§3.32) in the graph identified by the graphId property (§3.33.2) of the containing graphTraversal object (§3.33).

### message property

An edgeTraversal object **MAY** contain a property named message whose value is a message object (§3.11) that contains a message to display to the user as the edge is traversed.

### finalState property

An edgeTraversal object **MAY** contain a property named finalState whose value is an object (§3.6) each of whose properties represents the value of a relevant expression after the edge has been traversed. This property, together with graphTraversal.initialState (§3.33.4), enables a viewer to present a debugger-like “watch window” experience as the user traverses a graph.

For details of how properties within a “state” object are represented, see §3.37.6.

### stepOverEdgeCount property

An edgeTraversal object **MAY** contain a property named stepOverEdgeCount whose value is an integer specifying the number of edges a user can step over.

This property is intended to enable a viewing experience in which the user can either step over or step into the traversal of a nested graph (§3.31.5). Therefore, this property **SHOULD** be specified only on an edge that leads from a node to one of its child nodes, and its value **SHOULD** be the number of edges the user would need to traverse to return to the current nesting level.

If this property is present, a SARIF viewer **SHOULD** provide a visual cue informing the user that they have the option of either stepping over the current edge and into the nested graph, or of stepping over the entire traversal of the nested graph.

EXAMPLE: This example defines a graph containing two nested graphs, the first representing code locations in function A and the second representing locations in function B. Node na2 in function A represents a call to function B.

The example defines a graph traversal consisting of a set of edge traversals which start at node "na1" in function A, call into function B, and ultimately return to and continue execution in function A.

Suppose the user executes the first edge traversal, which traverses edge ea1.The next edge traversal has a stepOverEdgeCount property value of 4. Therefore, the SARIF viewer informs her that she can now choose to either step into function B by traversing edge "eab", or step over the function call by traversing 4 edges, the last of which (edge "eba") returns to function A at node "na3".

If she chooses to enter the nested graph, she will visit the following nodes, in this order:

[ na1, na2, nb1, nb2, nb3, na3, na4 ]

If she chooses not to enter the nested graph, the traversal of the edges

[ eab, eb1, eb2, eba ]

will be collapsed into a single “step over.” As a result, she will visit the following nodes, in this order:

[ na1, na2, na3, na4 ]

{ # A result object (§3.22).

"graphs": { # See §3.22.15.

"code": { # A graph object (§3.30).

"nodes": [

{

"id": "functionA",

"children": [

{ "id": "na1" },

{ "id": "na2", "label": "Call functionB" },

{ "id": "na3" },

{ "id": "na4" }

]

},

{

"id": "functionB",

"nodes": [

{ "id": "nb1" },

{ "id": "nb2" },

{ "id": "nb3" }

],

}

]

"edges": [

{ "id": "ea1", "sourceNodeId": "na1", "targetNodeId": "na2" },

{ "id": "ea2", "sourceNodeId": "na2", "targetNodeId": "na3" },

{ "id": "eab", "sourceNodeId": "na2", "targetNodeId": "nb1" },

{ "id": "ea3", "sourceNodeId": "na3", "targetNodeId": "na4" },

{ "id": "eb1", "sourceNodeId": "nb1", "targetNodeId": "nb2" },

{ "id": "eb2", "sourceNodeId": "nb2", "targetNodeId": "nb3" },

{ "id": "eba", "sourceNodeId": "nb3", "targetNodeId": "na3" }

]

}

},

"graphTraversals": [ # See §3.22.16.

{ # A graphTraversal object (§3.33).

"graphId": "code", # The graph being traversed.

"edgeTraversals": [

{ "edgeId": "ea1" },

{

"edgeId": "eab",

"stepOverEdgeCount": 4

},

{ "edgeId": "eb1" },

{ "edgeId": "eb2" },

{ "edgeId": "eba" },

{ "edgeId": "ea3" }

]

}

]

}

## 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 (§3.11) relevant to this call stack.

### frames property

A stack object **SHALL** contain a property named frames whose value is an array of zero or more stackFrame objects (§3.36). 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.

## stackFrame object

### General

A stackFrame object describes a single stack frame within a call stack (§3.35).

### location property

A stackFrame object **MAY** contain a property named location whose value is a location object (§3.23) 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.

### 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.byteOffset property (§3.25.11), if any, specified by the physicalLocation property (§3.36.2). physicalLocation.region.byteOffset 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 zero or more strings representing the parameters of the function call represented by this stack frame.

If parameters is absent, it **SHALL** default to an empty array.

## threadFlowLocation object

### General

A threadFlowLocation object represents a location visited by an analysis tool in the course of simulating or monitoring the execution of a program.

### location property

If location information is available, a threadFlowLocation object **SHALL** contain a property named location whose value is a location object (§3.23) that specifies the location to which the threadFlowLocation object refers. If location information is not available, location **SHALL** be absent.

There are analysis tools whose native output format includes the equivalent of a SARIF code flow, but which do not provide location information for every step in the code flow. A SARIF converter for such a format might not be able to populate location. However, if the native output format associates a human readable message with such a step, the SARIF converter **SHOULD** create a location object and populate only its message property (§3.23.6). A SARIF direct producer which creates such code flows **SHOULD** populate location.message, even if no actual location information is available.

EXAMPLE: In this example, a file is locked by another program before a thread attempts to write to it. The analysis tool has no location information for the other program; in fact, the analysis tool might merely be simulating an execution sequence in which a *hypothetical* external program locks the file. Nevertheless, it provides a helpful message.

Note the use of executionOrder (§3.37.8) to ensure that the location in the external program executes before the location in the program being analyzed.

{ # A codeFlow object (§3.28).

"threadFlows": [ # See §3.28.3.

{ # A threadFlow object (§3.29).

"message": { # See §3.29.3.

"text": "An external program."

},

"locations": [ # See §3.29.4.

{ # A threadFlowLocation object.

"executionOrder": 1,

"location": { # A location object with only a message.

"message": {

"text": "File is now locked."

}

}

}

]

},

{ # Another threadFlow object.

"message": {

"text": "The program being analyzed."

},

"locations": [

...

{

"executionOrder": 2,

"message": {

"text": "Attempt to write to the file."

},

"location": {

"physicalLocation": {

"fileLocation": {

"uri": "io/logger.c",

"uriBaseId": "SRCROOT"

},

"region": {

"startLine": 42,

"snippet": {

"text": " fprintf(fd, "test\\n");\n"

}

}

}

}

}

]

}

]

}

### module property

A threadFlowLocation 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.

### stack property

A threadFlowLocation object **MAY** contain a property named stack whose value is a stack object (§3.35) that represents the call stack leading to this location.

### kind property

A threadFlowLocation object **MAY** contain a property named kind whose value is a string that describes the meaning of this location. The interpretation of kind depends on the tool that produced the log file. A SARIF consumer that wishes to take action based on kind **SHALL** examine run.tool (§3.13.6, §3.16) to determine if it (the consumer) knows how to interpret the kind values produced by that tool.

kind **SHOULD** be a human-readable string (as opposed to, for example, a GUID or a hash value).

A SARIF producer **MAY** provide additional kind-dependent information by populating threadFlowLocation.properties with properties whose names and values depend on kind. A SARIF consumer that knows how to interpret kind for this tool **MAY** use this additional information.

EXAMPLE:

"kind": "taintedDataSource"

### state property

A threadFlowLocation object **MAY** contain a property named state whose value is an object (§3.6) in which each property name represents an expression relevant to the location in the context of the code flow, and the corresponding property value specifies the value of that expression. This property enables a SARIF viewer to present a debugger-like “watch window” experience as the user navigates through a code flow.

EXAMPLE 1: In this example, the state property captures the values of the expressions "x", "y", and "x + y".

{ # An threadFlowLocation object.

"state": {

"x": "42",

"y": "54",

"x + y": "96"

}

}

NOTE: A viewer might use these values to provide a “watch window” experience, showing the changing values of selected variables and expressions as the user steps through a code flow.

The format of each property name **SHALL** be consistent with the syntax of an expression in the programming language in which the code being analyzed was written. Each property value **SHALL** be a string whose format is consistent with the syntax of a value in the programming language in which the code being analyzed was written

EXAMPLE 2: In C++, a property name within the state object might be:

* A variable name such as "index".
* An array element reference such as "names[index]".
* An object property reference such as "names[index]->first".
* Any other expression that produces a value.

EXAMPLE 3: In C++, a property value within the state object might be:

* An integer such as "42" (note that the property value is a string).
* A string such as "\"John\"" (the double quotes are escaped as they would be in a JSON serialization; other serializations might represent the double quotes differently).
* A Boolean such as "true".

### nestingLevel property

A threadFlowLocation object **MAY** contain a property named nestingLevel whose value is an integer that represents any type of logical containment hierarchy among the threadFlowLocation objects in the threadFlow. Typically, it represents function call depth.

A viewer that renders a threadFlow **SHOULD** provide a visual representation of the value of nestingLevel. Typically, this would be an indentation indicating the depth of each location in the call tree.

### executionOrder property

A threadFlowLocation object **MAY** contain a property named executionOrder whose value is a positive integer that represents the temporal order in which execution reached this location, across all threadFlowLocation objects within all threadFlow objects belonging to a single codeFlow (§3.28). executionOrder values are assigned in increasing order of time; for example, execution reaches a threadFlowLocation whose executionOrder is 2 occurs before it reaches a threadFlowLocation whose executionOrder is 3. If two threadFlowLocations in different threadFlow objects within the same codeFlow have the same value for executionOrder, it means that execution reached both of those locations simultaneously. For that reason, values of executionOrder within a single threadFlow **SHALL** be unique.

It is only necessary to assign a value to executionOrder when the temporal ordering of a threadFlowLocation relative to a location in a different threadFlow is significant to the detection of a result.

If this property is absent, it **SHALL** default to 0, which is not otherwise a valid value for executionOrder.

### executionTimeUtc property

A threadFlowLocation object **MAY** contain a property named executionTimeUtc whose value is a string specifying the UTC date and time at which the thread of execution through the code reached this location. The string **SHALL** be in the format specified in §3.9.

### importance property

A threadFlowLocation **MAY** contain a property named importance whose value is a string that specifies the importance of this threadFlowLocation in understanding the code flow.

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.

## resultProvenance object

### General

A resultProvenance object contains information about the how and when its containing result was detected.

NOTE: This information is useful to various human and automated participants in an engineering system. For example:

* A build engineer might use the information to understand the specific tool invocation that produced the result, for example, if the violated rule should not have been configured to run at all.
* A developer reviewing results might use the information to determine how long an issue has existed in the code.
* A result management system might be responsible for associating logically identical results from one run to the next, making it possible for the developer to determine how long the result has existed. Such a result management system might populate this information.

### firstDetectionTimeUtc property

A resultProvenance object **MAY** contain a property named firstDetectionTimeUtc whose value is a string specifying the UTC date and time at which the result was first detected. The string **SHALL** be in the format specified in §3.9. It **SHOULD** specify the start time of the run in which the result was first detected, as opposed to, for example, the time within the run at which the result was actually generated.

NOTE: Using the run’s start time makes it possible to group together results that were first detected in the same run.

### lastDetectionTimeUtc property

A resultProvenance object **MAY** contain a property named lastDetectionTimeUtc whose value is a string specifying the UTC date and time at which the result was most recently detected. The string **SHALL** be in the format specified in §3.9. It **SHOULD** specify the start time of the run in which the result was most recently detected, as opposed to, for example, the time within the run at which the result was actually generated.

NOTE: Using the run’s start time makes it possible to group together results that were detected in the same run.

If lastDetectionTimeUtc is absent, its default value **SHALL** be determined as follows:

1. If run.invocations is present, and if the startTimeUtc property (§3.17.6) is present on any of the invocation objects (§3.17) in that array, then the default is the earliest of those times.
2. Otherwise, there is no default.

### firstDetectionRunInstanceGuid property

A resultProvenance object **MAY** contain a property named firstDetectionRunInstanceGuid whose value is a GUID-valued string (§3.5.3) which **SHALL** equal the id.instanceGuid property (§3.13.3, §3.15.5) of the run in which the containing result was first detected (either the current run or some previous run).

### lastDetectionRunInstanceGuid property

A resultProvenance object **MAY** contain a property named lastDetectionRunInstanceGuid whose value is a GUID-valued string (§3.5.3) which **SHALL** equal the id.instanceGuid property (§3.13.3, §3.15.5) of the run in which the containing result was most recently detected (either the current run or some previous run).

### invocationIndex property

If its containing run object (§3.13) has an invocations property (§3.13.7), a resultProvenance object **MAY** contain a property named invocationIndex whose value is a non-negative integer that specifies the index within the invocations array of the invocation object (§3.17) that describes the tool invocation as a result of which the containing result was detected.

If the containing run object does not have an invocations property, invocationIndex **SHALL** be absent.

NOTE 1: The purpose of this property is to allow a result to be associated with the tool invocation that produced it.

If invocationIndex is absent and run.invocations is present and contains a single element, it **SHALL** default to 0; otherwise it **SHALL** default to -1 (which is otherwise not a valid value for this property).

NOTE 2: This provides a sensible default in the common case where there is only a single tool invocation in the run.

### conversionSources property

Some analysis tools produce output files that describe the analysis run as a whole; we refer to these as “per-run” files. Some tools produce one or more output files for each result; we refer to these as “per-result” files. Some tools produce both per-run and per-result files.

A resultProvenance object **MAY** contain a property named conversionSources whose value is an array of one or more unique (§3.7.2) physicalLocation objects (§3.24).

If the result object (§3.22) containing this resultProvenance object was produced by a converter, and if the analysis tool whose output was converted to SARIF produced any per-result files for this result, then the physicalLocation objects in the array **SHALL** specify the relevant portions of the per-result files for this result.

Otherwise (that is, if the run object was not produced by a converter, or if there were no per-run files for this result), then if conversionProvenance is present, its value **SHALL** be an empty array.

If conversionSources is absent, it **SHALL** default to an empty array.

Per-run files are handled by the conversion.analysisToolLogFiles property (§3.19.4).

NOTE: 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 to 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 §3.18)

...

},

"results": [

{

"ruleId": "Assertions",

"message": {

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

},

...

"provenance": { # See §3.22.25.

"conversionSources": [ # An array of physicalLocation objects

{ # (§3.24).

"fileLocation": { # See §3.24.3.

"uri": "AndroidStudio.log",

"uriBaseId": "$LOGSROOT"

},

"region": { # See §3.24.4.

"startLine": 3,

"startColumn": 3,

"endLine": 12,

"endColumn": 13,

"snippet": {

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

}

}

}

],

...

}

}

]

}

]

}

## 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 an object (§3.6) each of whose properties represents a single localized string. The property names correspond to resource identifiers (§3.11.6) within message objects (§3.11). If the property name is used as the value of the messageId property (§3.11.9) of any message object in the containing run object (§3.13), the property value **SHALL** be a plain text string (§3.11.2). If the property name is used as the value of the richMessageId property (§3.11.10) of any message object in the containing run object, the property value **SHALL** be a rich text string (§3.11.3). 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 object.

EXAMPLE:

"resources": {

"messageStrings": {

"call": "Function call",

"return": "Function return"

}

}

### rules property

A resources object **MAY** contain a property named rules whose value is an array of zero or more unique (§3.7.2) rule objects (§3.40).

If rules is absent, it **SHALL** default to an empty array.

Some tools use the same identifier to refer to multiple distinct (although logically related) rules. Therefore the id properties (§3.40.3) of the rule objects do not need to be unique within the array.

EXAMPLE: In this example, two distinct but related rules have the same rule idThey are distinguished by their message strings.

"resources": {

"rules": [

{

"id": "CA1711",

"messageStrings": {

"default": "Rename type name {0} so that it does not end in '{1}'"

}

},

{

"id": "CA1711",

"messageStrings": {

"default": "Either replace the suffix '{0}' in member name '{1}' with

the suggested numeric alternate or provide

a more meaningful suffix"

}

}

]

}

## 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 (§3.40.6) or the fullDescription property (§3.40.7) 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.

### deprecatedIds property

A rule object **MAY** contain a property named deprecatedIds whose value is an array of zero or more unique (§3.7.2) strings, each of which contains a stable opaque identifier (that is, a rule id; see §3.40.3) by which this rule was known in some previous version of the analysis tool.

If deprecatedIds is absent, it **SHALL** default to an empty array.

NOTE: This property addresses the scenario where rule ids change from one version of a tool to the next. For example, a tool developer might decide that a rule is too general, covering too many concepts. In the next version of the tool, the tool developer might break this rule into a set of more specific rules.

Now the result management system has the problem of matching results between the newer and the older versions of the tool. deprecatedIds solves this problem.

EXAMPLE: In this example, version 1 of an analysis tool defines rule CA1000. A run of this tool finds two results.The result management system decides that neither result was previously detected, so it marks them as with "baselineState": "new" (§3.22.20), producing this log:

{

"tool": {

"name": "CodeScanner",

"version": "1"

},

"results": [

{

"ruleId": "CA1000",

"ruleIndex": 0,

"baselineState": "new",

...

},

{

"ruleId": "CA1000",

"ruleIndex": 0,

"baselineState": "new",

...

}

],

"resources": {

"rules": [

{

"id": "CA1000",

...

}

]

}

}

The engineering team decides that these results are false positive, so they add in-source suppressions, for example (in C#):

[SuppressMessage("CA1000", ...)]

...

[SuppressMessage("CA1000", ...)]

Now the tool developers decide that rule CA1000 is too broad, so in version 2 of the tool, they divide it into two new rules, CA1001 and CA1002. The engineering team runs the new tool, and the result management system performs result matching, producing this log:

{

"tool": {

"name": "CodeScanner",

"version": "2"

},

"results": [

{

"ruleId": "CA1001",

"ruleIndex": 0,

"baselineState": "existing",

"suppressionStates": [

"suppressedInSource"

],

...

},

{

"ruleId": "CA1002",

"ruleIndex": 0

"baselineState": "existing",

"suppressionStates": [

"suppressedInSource"

],

...

}

],

"resources": {

"rules": [

{

"id": "CA1001",

"deprecatedIds": [

"CA1000"

],

...

},

{

"id": "CA1002",

"deprecatedIds": [

"CA1000"

],

...

}

]

}

}

There are a few things to notice:

* In resources.rules, each of the new rules is associated with its id from the previous tool version.
* As a result, the analysis tool can determine that the in-source suppressions still apply, even though the rule ids have changed, so it correctly marks each result with "suppresionStates": [ "suppressedInSource" ].
* Furthermore, the result management system can determine that these are the same results it saw in the previous run, so it correctly marks them with "baselineState": "existing".

### name property

A rule object **MAY** contain a property named name whose value is a message object (§3.11) 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 (§3.11) 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 (§3.11) 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 (§3.40.6). 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 an object (§3.6) consisting of a set of properties with arbitrary names.

The value of each property **SHALL** be a plain text message string (§3.11.2). As with any message string, it **MAY** contain placeholders (§3.11.4) and embedded links (§3.11.5).

The set of property names appearing in the messageStrings property **SHALL** contain at least the set of strings which occur as values of result.message.messageId properties (§3.22.9) in the current run object. The messageStrings property **MAY** contain additional properties whose names do not appear as the value of the result.message.messageId 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 (§3.40.8), it **MAY** also contain a property named richMessageStrings whose value is an object (§3.6) consisting of a set of properties each of whose names also appears as a property name in messageStrings (§3.40.8).

The value of each property **SHALL** be a rich text message string (§3.11.3). As with any message string, it **MAY** contain placeholders (§3.11.4) and embedded links (§3.11.5).

SARIF consumers that cannot render rich text **SHALL** ignore richMessageStrings and use messageStrings instead.

NOTE: This is the reason for the requirement that every property name that appears in richMessageStrings also appears in messageStrings.

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 §3.11.3.2.

### helpUri property

A rule object **MAY** contain a property named helpUri whose value is a string containing the absolute URI [[RFC3986](#RFC3986)] of the primary documentation for the rule.

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 (§3.11) 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 (§3.41).

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 §3.41.

The configuration parameters for a rule, if any, **SHALL NOT** be stored in the rule object’s property bag (§3.8) Rather, they **SHALL** be stored in ruleConfiguration.parameters (§3.41.5).

## 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 §3.41.5.

### enabled property

A ruleConfiguration object **MAY** contain a property named enabled whose value is a Boolean that specifies whether the rule was evaluated during the scan.

If this property is absent, it **SHALL** default to 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", or "note", with the same meanings as when those strings appear as the value of result.level (§3.22.8).

If defaultLevel is absent, it **SHALL** default to "warning".

If defaultLevel is present, it **SHALL** provide the value for the level property of any result object (§3.22) whose ruleIndex property (§3.22.6) refers to the containing rule object (§3.40) 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"

### defaultRank property

A ruleConfiguration object **MAY** contain a property named defaultRank whose value is a number between 0.0 and 100.0 inclusive, with the same interpretation as the value of the result.rank (§3.22.21).

If defaultRank is absent, it **SHALL** default to -1.0 (which is otherwise not a valid value for this property).

If defaultRank is present, it **SHALL** provide the value for the rank property of any result object (§3.22) whose ruleIndex property (§3.22.6) refers to the containing rule object (§3.40) and which does not itself specify a rank property.

### parameters property

A ruleConfiguration object **MAY** contain a property named parameters whose value is a property bag (§3.8). 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 maximum length.

{ # A rule object (§3.40).

"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 containing result object (§3.22). 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 (§3.22).

"fixes": [ # See §3.22.26.

{ # A fix object.

"description": { # See §3.42.2.

"text": "Private member names begin with '\_'"

},

"fileChanges": [ # See §3.42.3.

{ # A fileChange object (§3.43).

...

}

]

}

],

...

}

### description property

A fix object **SHOULD** contain a property named description whose value is a message object (§3.11) 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 unique (§3.7.2) fileChange objects (§3.43), each of which describes the changes to a single file that are necessary to effect the fix.

NOTE: fileChanges is an array because a fix might require changes to multiple files.

The array elements **SHALL** refer to distinct files.

If fileChanges is absent, it **SHALL** default to an empty array.

EXAMPLE 1: In this example, two fileChange objects make identical changes (commenting out the first line) in two distinct C-language files, src/a.c and src/b.c.

{ # A fix object.

"fileChanges": [

{ # A fileChange object (§3.43).

"fileLocation": { # See §3.43.2.

"uri": "src/a.c"

},

"replacements": [ # See §3.43.3.

{ # A replacement object (§3.44).

"deletedRegion": { # See §3.44.3.

"startLine": 1,

"startColumn": 1,

"endColumn": 1

},

"insertedContent": { # See §3.44.4.

"text": "// "

}

}

}

},

{

"fileLocation": {

"uri": "src/b.c"

},

"replacements": [

{

"deletedRegion": {

"startLine": 1,

"startColumn": 1,

"endColumn": 1

},

"insertedContent": {

"text": "// "

}

}

}

}

]

}

EXAMPLE 2: This example represents invalid SARIF because the two fileChange objects refer to the same file, src/a.c. It is invalid even though the fileChange objects are distinguished by their replacements properties.

{ # A fix object.

"fileChanges": [

{ # A fileChange object (§3.43).

"fileLocation": { # See §3.43.2.

"uri": "src/a.c"

},

"replacements": [ # See §3.43.3.

{ # A replacement object (§3.44).

"deletedRegion": { # See §3.44.3.

"startLine": 1,

"startColumn": 1,

"endColumn": 1

},

"insertedContent": { # See §3.44.4.

"text": "// "

}

}

}

},

{

"fileLocation": {

"uri": "src/a.c" # Invalid: refers to the same file.

},

"replacements": [

{

"deletedRegion": {

"startLine": 2, # Invalid even though it affects a

"startColumn": 1, # different line.

"endColumn": 1

},

"insertedContent": {

"text": "// "

}

}

}

}

]

}

## fileChange object

### General

A fileChange object represents a change to a single file.

EXAMPLE:

{ # A fix object (§3.41).

"fileChanges": [ # See §3.42.3.

{

"fileLocation": { # See §3.43.2.

"uri": "a.h"

},

"replacements": [ # See §3.43.3.

{ # A replacement object (§3.44).

...

},

{ # Another replacement object.

...

}

]

}

]

}

### fileLocation property

A fileChange object **SHALL** contain a property named fileLocation whose value is a fileLocation object (§3.4) 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 (§3.44), each of which represents the replacement of a single region of the file specified by the fileLocation property (§3.43.2).

## 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 (§3.44.3) and the insertion of new file content by means of the insertedContent property (§3.44.4), then the effect of the replacement **SHALL** be as if the removal were performed before the insertion.

If a single fileChange object (§3.43) 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 (§3.43.3). 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": {

"byteOffset": 12,

"byteLength": 5

},

"insertedContent": {

"binary": "ZXhhbXBsZQ=="

}

},

{

"deletedRegion": {

"byteOffset": 20,

"byteLength": 3

}

},

{

"deletedRegion": {

"byteOffset": 312,

"byteLength": 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 of the file contents after the first change. 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 (§3.44.3) specifies a text region (§3.25.2) or a binary region (§3.25.3).

EXAMPLE 2: In this example, the replacements property specifies a replacement in a text file.

"replacements": [

{

"deletedRegion": { # The region object represents a text region (§3.25.2).

"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 (§3.44.3) specifies a text region (§3.25.2) and the insertedContent property (§3.44.4) is present, then the insertedContent property **SHOULD** contain a text property (§3.3.2).

If the deletedRegion property specifies a binary region (§3.25.3) and the insertedContent property is present, then the insertedContent property **SHALL** contain a binary property (§3.3.3).

### deletedRegion property

A replacement object **SHALL** contain a property named deletedRegion whose value is a region object (§3.25) 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 (§3.3) 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 the inserted content is specified as text, the text **SHALL** be transcoded from UTF-8 (the encoding of all text in all SARIF log files) to the encoding of the target file before being inserted.

NOTE: This implies that a text fix cannot be safely applied unless the target file’s encoding is known.

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 (§3.22).

### 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, §3.40.3), which must be stable and opaque, notification identifiers do not 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 (§3.40.3).

### ruleIndex property

Depending on the circumstances, a notification object either **SHOULD** or **SHALL NOT** contain a property named ruleIndex whose value is a non-negative integer that specifies the index of the relevant rule object (§3.40) in the resources.rules property (§3.13.15, §3.39.3) of the containing run object (§3.13), or in an external resource file (§3.11.6.4) or external property file (§3.13.2.1). If this property is absent, it **SHALL** default to -1 (which is otherwise not a valid value for this property).

NOTE: If ruleIndex is absent, a SARIF consumer will not be able to locate metadata for the rule.

If the relevant rule object is present in resources.rules or in an external resource file or external property file, then ruleIndex **SHOULD** be present. Otherwise, ruleIndex **SHALL NOT** be present.

EXAMPLE: In this example, there is more than one rule with id CA1711. ruleIndex uniquely specifies the relevant rule, whether or not there are multiple rules with the same id.

{ # A run object (§3.13).

"invocations": [ # See §3.13.7.

{ # An invocation object (§3.17).

"configurationNotifications": [ # See §3.17.21.

{ # A notification object.

"id": "CFG0001",

"message": {

"text": "Rule configuration is missing."

},

"ruleId": "CA1711",

"ruleIndex": 1

}

],

}

],

"resources": { # See §3.13.15.

"rules": [ # See §3.39.3.

{ # A rule object (§3.40).

"id": "CA1711",

...

},

{ # Another rule object with the same id.

"id": "CA1711",

...

}

}

}

}

If ruleIndex and ruleId are both present, then the value of the id property (§3.40.3) of the rule object specified by ruleIndex either **SHALL** equal the value of ruleId , or it **SHALL** equal a substring of ruleId that starts at the beginning of the string and ends before the last forward slash. For an example, see Example 2 in §3.22.6.

### 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 (§3.24) that identifies the relevant location.

### message property

A notification object **SHALL** contain a property named message whose value is a message object (§3.11) that describes the condition that was encountered.

NOTE: The message object in the notification.message property will typically not contain a richText (§3.11.8) or richMessageId (§3.11.10) 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.

If level is absent, it **SHALL** default to "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.

### timeUtc property

A notification object **MAY** contain a property named timeUtc whose value is a string specifying the UTC date and time at which the analysis tool generated the notification. The string **SHALL** be in the format specified by (§3.9).

### 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 (§3.46).

If the notification is not the result of a runtime exception, the exception property **SHALL** be absent.

## 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 message object (§3.11) 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 Message property of any object derived from System.Exception.

NOTE: Exception messages, appearing as they do in typical languages and operating systems, are typically plain text, and require no arguments (§3.11.4). Nevertheless, this property is defined to be a message object rather than a simple string for consistency with other message properties in this specification.

### stack property

An exception object **MAY** contain a property named stack whose value is a stack object (§3.35) 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 zero or more exception objects, each of which is considered a cause of the containing exception.

If innerExceptions is empty, it **SHALL** default to an empty array.

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.

# External property file format

## General

External property files (see §3.13.2) conform to a schema distinct from that of the root file. External property files contain information that makes it possible for a consumer to determine which properties are contained in the file, to parse their contents, and to associate the external properties with the run to which they belong.

## External property file naming convention

The file name of an external property file **SHOULD** end with the extension ".sarif-external-properties".

EXAMPLE 1: scan-results.sarif-external-properties

The file name **MAY** end with the additional extension ".json".

EXAMPLE 2: scan-results.sarif-external-properties.json

## externalProperties object

### General

The top-level element of an external property file **SHALL** be an object which we refer to as an externalProperties object.

EXAMPLE: In this example, the files property and the properties property have been externalized to a file with these contents:

{ # An externalProperties object

"$schema": # See §4.3.2.

"http:///json.schemastore.org/sarif-external-property-file-2.0.0",

"version": "2.0.0", # See §4.3.3.

# See §4.3.4.

"instanceGuid": "00001111-2222-3333-4444-555566667777",

# See §4.3.5.

"runInstanceGuid": "88889999-AAAA-BBBB-CCCC-DDDDEEEEFFFF",

"run.files": { # See §4.3.6.

"apple.png": {

"mimeType": "image/png"

},

"banana.png": {

"mimeType": "image/png"

}

},

"run.properties": {

"policy": "Web application security policy",

"team": "Security Assurance Team"

}

}

### $schema property

An externalProperties object **MAY** contain a property named $schema whose value is a string containing an absolute URI from which a JSON schema document describing the version of the external property file format to which this external property 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 external property file format corresponding to the SARIF version specified by the version property (§4.3.3).

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

### version property

An externalProperties object **SHALL** contain a property named version whose value is a string designating the version of the SARIF specification to which this external property 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 external property file format if new versions are defined in the future.

### instanceGuid property

An externalProperties object **SHALL** contain a property named instanceGuid whose value is a GUID-valued string (§3.5.3) that equals the instanceGuid property (§3.14.3) of the corresponding externalPropertyFile object (§3.14) in the run.externalPropertyFiles property (§3.13.2) in the root file.

### runInstanceGuid property

If the run object to which the external property belongs contains a id.instanceGuid property (§3.13.3, §3.15.5), the externalProperties object **MAY** contain a property named runInstanceGuid whose value is a GUID-valued string (§3.5.3) that equals id.instanceGuid. If the run object does not define id.instanceGuid, then runInstanceGuid **SHALL** be absent.

### The property value properties

An externalProperties object **SHALL** contain one or more properties each of whose names is constructed by prepending the string "run." to the name of the external property and whose corresponding values are the values of the external properties, exactly as those properties would have appeared had they occurred inline in the root file.

NOTE: See the EXAMPLE in §4.3.1, where the names of the external properties are files and properties, the externalized value of the files property is stored in a property named run.files, and the externalized value of the properties property is stored in a property named run.properties.

# 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**: A log file in the format defined by the SARIF specification.
* **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.
* **Deterministic producer**: A SARIF producer which, given identical inputs, repeatedly produces an identical SARIF log file.
* **Converter**: A SARIF producer that transforms the output of an analysis tool from its native output format into the SARIF format.
* **SARIF post-processor**: A SARIF producer that transforms an existing SARIF log file into a new SARIF log file, for example, by removing or redacting security-sensitive elements.
* **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.
* **Result management system**: a software system that consumes the log files produced by analysis tools, produces reports that enable engineering teams to assess the quality of their software artifacts 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.
* **Engineering system**: a software development environment within which analysis tools execute. It might include a build system, a source control system, a [result management system](#def_result_management_system), a bug tracking system, a test execution system, and so on.

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 another conformance target.

## 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 §3.11.6.5, “SARIF resource file format”.
* It contains only those elements defined in §3.11.6.5.
* Those elements that it does contain conform to the syntax and semantics defined in §3, except as modified in §3.11.6.5.

## 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: 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](#AppendixDeterminism), “Producing deterministic SARIF log files”.

## Conformance Clause 6: Converter

A converter satisfies the “Converter” conformance profile if:

* It satisfies the “SARIF producer” conformance profile.
* It additionally 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 7: SARIF post-processor

A SARIF post-processor satisfies the “SARIF post-processor” conformance profile if:

* It satisfies the “SARIF producer” conformance profile.
* It additionally satisfies those normative requirements in §3 that are designated as applying to post-processors.

## Conformance Clause 8: 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 9: 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.

## Conformance Clause 10: Result management system

A result management system satisfies the “result management system” conformance profile if:

* It satisfies the “SARIF consumer” conformance profile.
* It additionally satisfies the normative requirements in §3 and [Appendix B](#AppendixFingerprints) (“Use of fingerprints by result management systems”) that are designated as applying to result management systems.

## Conformance Clause 11: Engineering system

An engineering system satisfies the “engineering system” conformance profile if:

* It satisfies the normative requirements in §3 that are designated as applying to engineering systems.

1. (Informative) Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Andrew Pardoe, Microsoft

Chris Wysopal, CA Technologies

David Keaton, Individual

Douglas Smith, Kestrel Technology

Duncan Sparrell, sFractal Consulting LLC

Everett Maus, Microsoft

Hendrik Buchwald, RIPS Technologies

Henny Sipma, Kestrel Technology

Jim Kupsch, SWAMP

Jordyn Puryear, Microsoft

Joseph Feiman, CA Technologies

Ken Prole, Code Dx, Inc.

Kevin Greene, Mitre Corporation

Larry Hines, Micro Focus

Laurence J. Golding, Individual

Luke Cartey, Semmle

Mel Llaguno, Synopsys

Michael Fanning, Microsoft

Nikolai Mansourov, Object Management Group

Paul Anderson, GrammaTech, Inc.

Paul Brookes, Microsoft

Paul Patrick, FireEye, Inc.

Philip Royer, Splunk Inc.

Pooya Mehregan, Security Compass

Ram Jeyaraman, Microsoft

Sean Barnum, FireEye, Inc.

Smith Douglas, Kestrel Technology

Stefan Hagen, Individual

Sunny Chatterjee, Microsoft

Tim Hudson, Cryptsoft Pty Ltd.

Trey Darley, New Context Services, Inc.

Vamshi Basupalli, SWAMP

Yekaterina O'Neil, Micro Focus

1. (Normative) 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 engineering 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 logically 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 **SHOULD** 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 result.message, for example: "The word xxx should not be used in documentation."

The SARIF format provides the partialFingerprints 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 partialFingerprints object to the prohibited word. A result management system **SHALL** include the information in partialFingerprints in its fingerprint computation. See §3.22.13 for more requirements on how a result management system decides which partial fingerprints to use.

An analysis tool **SHALL NOT** include in partialFingerprints information that a result management system could deduce from other information in the SARIF file, for example, file hashes. Rather, the result management would use such information, along with partialFingerprints, in its computation of fingerprints.

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.

A result management system **SHOULD NOT** include an absolute line number (or an absolute byte location in a binary file) in its fingerprint computation.

A result management system **SHALL NOT** include non-deterministic file format elements ([Appendix F](#AppendixDeterminism), §F.2) in its fingerprint computation.

A result management system **SHALL NOT** include non-deterministic absolute URIs ([Appendix F](#AppendixDeterminism), §F.4) in its fingerprint computation.

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.physicalLocation property, but should not attempt to populate result.analysisTarget (see §3.22.11).
* 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 §3.16.4).

1. (Informative) Locating rule metadata

The SARIF format allows rule metadata to be included in a SARIF log file (see §3.13.15 and §3.39). 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.startTimeUtc
* invocation.endTimeUtc
* 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)
* threadFlow.threadId
* threadFlowLocation.executionTimeUtc
* notification.threadId
* notification.timeUtc
* result.instanceGuid
* run.id.instanceId
* run.id.instanceGuid
* run.aggregateIds[i].instanceId
* run.aggregateIds[i].instanceGuid
* run.baselineInstanceGuid
* 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, SARIF requires a specific ordering. For example, within stack.Frames, SARIF requires the location object representing the most deeply nested function call to appear first.

For other arrays, for example properties.tags, SARIF does not require a specific ordering. 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 tags 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 non-deterministic absolute file paths (that is, absolute paths which might differ from machine to machine) in fileLocation.uri properties prevents the production of 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 non-deterministic 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 it 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 file content. 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 replacements maximizes the amount of work that can be done by automated fixup tools.

1. (Informative) Diagnosing results in generated files

Sometimes it is desirable to analyze files generated by the build. These files are usually not under source control, and the build might even overwrite them multiple times. This Appendix offers guidance on how to persist enough information in a SARIF log file to facilitate the diagnosis of results in these files.

In what follows, we will refer to files that are generated only once as “singly generated,” and files that are generated multiple times as “multiply generated.”

It can be difficult to diagnose results in generated files for the following reasons:

* The file might not be available to the engineer who diagnoses the result (for example, the engineer might not have a build environment).
* If the file is multiply generated, then at best only the last version is available, but results might have been found in previous versions.
* It might be difficult to tell which instance of a multiply generated file contained the result.

For both singly and multiply generated files, there are two options (which can be used together):

1. Use the physicalLocation object’s region and contextRegion properties to store enough of the generated file’s contents to facilitate diagnosis. The region object’s snippet property holds the relevant portion of the file contents.
2. Use the file object’s contents property to persist the entire contents of the file in run.files.

The first option is more compact; the second allows a SARIF viewer to present results with greater context.

EXAMPLE 1: In this example, the analysis tool populates region.snippet and contextRegion.snippet, allowing a SARIF viewer to display just enough context (one hopes) to diagnose the result.

{ # A run object (§3.13).

"originalUriBaseIds": { # See §3.13.10

"GENERATED": "file:///C:/code/browser/obj"

},

"results": [ # See §3.13.14.

{ # A result object (§3.22).

"ruleId": "CS6789", # See §3.22.5.

"message": { # See §3.22.9.

"text": "Division by 0!"

},

"locations": [ # See §3.22.10.

{ # A location object (§3.23).

"physicalLocation": { # See §3.23.3.

"fileLocation": {

"uri": "ui/window.g.cs", # A generated file (".g").

"uriBaseId": "GENERATED"

},

"region": {

"startLine": 42,

**"snippet": {**

**"text": " int z = x / y;\r\n"**

**}**

},

"contextRegion": {

"startLine": 40,

"endLine": 42,

**"snippet": {**

**"text": " int x = 54;\r\n int y = 0;\r\n int z = x / y;\r\n"**

**}**

}

}

}

]

}

],

...

}

EXAMPLE 2: In this example, the analysis tool populates file.contents, allowing a SARIF viewer to present the result in a larger context at the expense of a larger log file.

{

"originalUriBaseIds": {

"GENERATED": {

"uri": "file:///dev-1.example.com/code/browser/obj/"

}

},

"results": [

{

"ruleId": "CS6789",

"message": {

"text": "Division by 0!"

},

"locations": [

{

"physicalLocation": {

"fileLocation": {

"uri": "ui/window.g.cs",

"uriBaseId": "GENERATED",

"fileIndex": 0

},

"region": {

"startLine": 42

},

"contextRegion": {

"startLine": 40,

"endLine": 42

}

}

}

]

}

],

"files": [ # See §3.13.11.

{ # A file object (§3.21).

"fileLocation": { # See §3.21.2.

"uri": "ui/window.g.cs",

"uriBaseId": "GENERATED"

},

**"contents": { # See §3.21.8.**

**"text": "..." # See §3.3.2.**

**}**

}

]

}

Multiply generated files are treated similarly, but they present an additional problem: if more than one version of a given multiply generated file appears in run.files – either because the analysis tool wishes to persist the file contents, or for any other reason – then there must be a way to distinguish them.

The recommended solution is for the analysis tool to create a new URI base id for each version of the generated files. For example, the tool might append an incremented integer to the URI base id for each version of the file. The result might look like the following example.

EXAMPLE 3: In this example, "ui/window.g.cs" is multiply generated. The analysis tool creates URI base ids "GENERATED-1" and "GENERATED-2" to distinguish the two versions.

{

"originalUriBaseIds": {

**"GENERATED-1": {**

**"uri": "file:///dev-1.example.com/code/browser/obj/"**

**},**

**"GENERATED-2": {**

**"uri": "file:///dev-1.example.com/code/browser/obj/"**

**}**

},

"results": [

{

"ruleId": "CS6789",

"message": {

"text": "Division by 0!"

},

"locations": [

{

"physicalLocation": {

"fileLocation": {

"uri": "ui/window.g.cs",

**"uriBaseId": "GENERATED-1"**

},

"region": {

"startLine": 42

},

"contextRegion": {

"startLine": 40,

"endLine": 42

}

}

}

]

}

],

"files": [

"fileLocation": {

"uri": "ui/window.g.cs",

**"uriBaseId": "GENERATED-1" # Unique uriBaseId value.**

}

...

},

{

"fileLocation": {

"uri": ui/window.g.cs",

**"uriBaseId": "GENERATED-2" # Unique uriBaseId value.**

}

...

}

]

}

1. (Informative) Sample sourceLanguage values

This Appendix contains a list of sample values for the file.sourceLanguage property (§3.21.10) for some common programming languages. The purpose of this Appendix is to promote interoperability by encouraging SARIF producers to use the same identifiers for these languages.

The names of some of the languages in this list are the trademarks of their respective owners.

* abap
* actionscript
* apex
* c
* cobol
* coldfusion
* cplusplus
* csharp
* css
* fsharp
* fortran
* go
* java
* javascript
* json
* jsp
* markdown (variants: markdown/gfm, markdown/cmark)
* objectivec
* perl
* php
* python
* razor
* ruby
* scala
* sql (variants: sql/tsql, sql/psql).
* swift
* typescript
* yaml
* Markup languages:
  + html
  + sgml
  + xml
* Typesetting languages:
  + latex
  + nroff
  + roff
  + tex
  + troff
* Unix ® shell languages:
  + bash
  + csh
  + ksh
  + sh
  + tcsh
* Windows ® shell languages:
  + cmd
  + powershell

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 (elements which the specification states **SHALL** be present).

The file contains a single run object (§3.13) with an empty results array (§3.13.14), as would happen if the tool detected no issues in any of the files it scanned.

{

"version": "2.0.0",

"runs": [

{

"tool": {

"name": "CodeScanner"

},

"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 §3.13.14), and
2. The analysis tool has source location information available.

The file contains those elements recommended by the specification (elements which the specification states “**SHOULD**” be present), in addition to the required elements.

The file contains a single run object (§3.13) with a results array (§3.13.14). The results array contains a single result object (§3.22) so the recommended elements of the result object can be shown.

Its run.files property (§3.13.11) specifies only those files in which the tool detected a result.

It does not contain a run.logicalLocations property (§3.13.12), because when physical location information is available, that property is optional (it “**MAY**” be present).

This example also includes a run.resources.rules property (§3.13.15, §3.39.3) 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"

},

"files": [

{

"fileLocation": {

"uri": "file:///build.example.com/work/src/collections/list.cpp"

},

"mimeType": "text/x-c"

}

],

"results": [

{

"ruleId": "C2001",

"ruleIndex": 0,

"message": {

"text": "Variable \"count\" was used without being initialized.",

"richText": "Variable `count` was used without being initialized."

},

"locations": [

{

"physicalLocation": {

"fileLocation": {

"uri": "file://build.example.com/work/src/collections/list.cpp",

"fileIndex": 0

},

"region": {

"startLine": 15

}

},

"fullyQualifiedLogicalName": "collections::list:add"

}

]

}

],

"resources": {

"rules": [

{

"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 §3.13.14), but
2. The analysis tool does not have source location information available.

The file contains those elements recommended by the specification (elements which the specification states “**SHOULD**” be present), in addition to the required elements.

The file contains a single run object (§3.13) with a results array (§3.13.14). The results array contains a single result object (§3.22) so the recommended elements of the result object can be shown.

Its run.files property (§3.13.11) specifies only those files in which the tool detected a result.

It contains a run.logicalLocations property (§3.13.12), because when physical location information is not available, that property is recommended.

{

"version": "2.0.0",

"runs": [

{

"tool": {

"name": "BinaryScanner"

},

"files": [

{

"fileLocation": {

"uri": "file://build.example.com/work/bin/example"

},

"mimeType": "application/vnd.microsoft.portable-executable"

}

],

"logicalLocations": [

{

"name": "Example",

"kind": "namespace"

},

{

"name": "Worker",

"fullyQualifiedName": "Example.Worker",

"kind": "type",

"parentIndex": 0

},

{

"name": "DoWork",

"fullyQualifiedName": "Example.Worker.DoWork",

"kind": "function",

"parentIndex": 1

}

],

"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",

"logicalLocationIndex": 2

}

]

}

]

}

]

}

* 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 (§3.11.6.5) and contains rule metadata for the language specified by tool.language (§3.16.8).

{

"version": "2.0.0",

"runs": [

{

"tool": {

"name": "BinaryAnalyzer",

"language": "en-US"

},

"resources": {

"rules": [

{

"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}'."

},

"configuration": {

"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": {

"instanceGuid": "BC650830-A9FE-44CB-8818-AD6C387279A0",

"instanceId": "Nightly code scan/2018-10-08"

},

"baselineInstanceGuid": "0A106451-C9B1-4309-A7EE-06988B95F723",

"aggregateIds": [

{

"instanceId": "Build/14.0.1.2/Release/20160716-13:22:18",

"correlationGuid": "26F138B6-6014-4D3D-B174-6E1ACE9439F3"

}

],

"tool": {

"name": "CodeScanner",

"fullName": "CodeScanner 1.1 for Microsoft Windows (R) (en-US)",

"version": "2.1",

"semanticVersion": "2.1.0",

"dottedQuadFileVersion": "2.1.0.0",

"language": "en-US",

"sarifLoggerVersion": "1.25.0",

"properties": {

"copyright": "Copyright (c) 2017 by Example Corporation.

All rights reserved."

}

},

"versionControlProvenance": [

{

"repositoryUri": "https://github.com/example-corp/browser",

"revisionId": "de67ef7",

"mappedTo": {

"uriBaseId": "PROJECTROOT"

}

}

],

"originalUriBaseIds": {

"PROJECTROOT": {

"uri": "file://build.example.com/work/"

},

"SRCROOT": {

"uri": " src/",

"uriBaseId": "PROJECTROOT"

},

"BINROOT": {

"uri": " bin/",

"uriBaseId": "PROJECTROOT"

}

},

"invocations": [

{

"commandLine": "CodeScanner @build/collections.rsp",

"responseFiles": [

{

"uri": "build/collections.rsp",

"uriBaseId": "SRCROOT",

"fileIndex": 0

}

],

"startTimeUtc": "2016-07-16T14:18:25Z",

"endTimeUtc": "2016-07-16T14:19:01Z",

"machine": "BLD01",

"account": "buildAgent",

"processId": 1218,

"fileName": "/bin/tools/CodeScanner",

"workingDirectory": {

"uri": "/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": "C2001",

"ruleIndex": 0,

"level": "error",

"message": {

"text": "Exception evaluating rule \"C2001\". Rule disabled;

run continues."

},

"physicalLocation": {

"fileLocation": {

"uri": "crypto/hash.cpp",

"uriBaseId": "SRCROOT",

"fileIndex": 4

}

},

"threadId": 52,

"timeUtc": "2016-07-16T14:18:43.119Z",

"exception": {

"kind": "ExecutionEngine.RuleFailureException",

"message": {

"text": "Unhandled exception during rule evaluation."

},

"stack": {

"frames": [

{

"location": {

"message": {

"text": "Exception thrown"

},

"fullyQualifiedLogicalName":

"Rules.SecureHashAlgorithmRule.Evaluate"

},

"module": "RuleLibrary",

"threadId": 52,

"address": 10092852

},

{

"location": {

"fullyQualifiedLogicalName":

"ExecutionEngine.Engine.EvaluateRule"

},

"module": "ExecutionEngine",

"threadId": 52,

"address": 10073356

}

]

},

"innerExceptions": [

{

"kind": "System.ArgumentException",

"message": "length is < 0"

}

]

}

},

{

"id": "CTN0002",

"level": "note",

"message": {

"text": "Run ended."

}

}

]

}

],

"files": [

{

"fileLocation": {

"uri": "build/collections.rsp",

"uriBaseId": "SRCROOT"

},

"mimeType": "text/plain",

"length": 81,

"contents": {

"text": "-input src/collections/\*.cpp -log out/collections.sarif -rules all -disable C9999"

}

},

{

"fileLocation": {

"uri": "application/main.cpp",

"uriBaseId": "SRCROOT"

},

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

"length": 1742,

"hashes": {

"sha-256": "cc8e6a99f3eff00adc649fee132ba80fe333ea5a"

}

},

{

"fileLocation": {

"uri": "collections/list.cpp",

"uriBaseId": "SRCROOT"

},

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

"length": 980,

"hashes": {

"sha-256": "b13ce2678a8807ba0765ab94a0ecd394f869bc81"

}

},

{

"fileLocation": {

"uri": "collections/list.h",

"uriBaseId": "SRCROOT"

},

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

"length": 24656,

"hashes": {

"sha-256": "849be119aaba4e9f88921a99e3036fb6c2a8144a"

}

},

{

"fileLocation": {

"uri": "crypto/hash.cpp",

"uriBaseId": "SRCROOT"

},

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

"length": 1424,

"hashes": {

"sha-256": "3ffe2b77dz255cdf95f97d986d7a6ad8f287eaed"

}

},

{

"fileLocation": {

"uri": "app.zip",

"uriBaseId": "BINROOT"

},

"mimeType": "application/zip",

"length": 310450,

"hashes": {

"sha-256": "df18a5e74b6b46ddaa23ad7271ee2b7c5731cbe1"

}

},

{

"fileLocation": {

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

},

"mimeType":

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

"parentIndex": 5,

"offset": 17522,

"length": 4050

}

],

"logicalLocations": [

{

"name": "add",

"fullyQualifiedName": "collections::list::add",

"decoratedName": "?add@list@collections@@QAEXH@Z",

"kind": "function",

"parentIndex": 1

},

{

"name": "list",

"fullyQualifiedName": "collections::list",

"kind": "type",

"parentIndex": 2

},

{

"name": "collections",

"kind": "namespace"

},

{

"name": "add\_core",

"fullyQualfiedName": "collections::list::add\_core",

"decoratedName": "?add\_core@list@collections@@QAEXH@Z",

"kind": "function",

"parentIndex": 1

},

{

"fullyQualifiedName": "main",

"kind": "function"

}

],

"results": [

{

"ruleId": "C2001",

"ruleIndex": 0,

"kind": "fail",

"level": "error",

"message": {

"messageId": "default",

"arguments": [

"ptr"

]

},

"suppressionStates": [ "suppressedExternally" ],

"baselineState": "existing",

"rank": 95,

"analysisTarget": {

"uri": "collections/list.cpp",

"uriBaseId": "SRCROOT",

"fileIndex": 2

},

"locations": [

{

"physicalLocation": {

"fileLocation": {

"uri": "collections/list.h",

"uriBaseId": "SRCROOT",

"fileIndex": 3

},

"region": {

"startLine": 15,

"startColumn": 9,

"endLine": 15,

"endColumn": 10,

"charLength": 1,

"charOffset": 254,

"snippet": {

"text": "add\_core(ptr, offset, val);\n return;"

}

}

},

"fullyQualifiedLogicalName": "collections::list:add",

"logicalLocationIndex": 0

}

],

"relatedLocations": [

{

"message": {

"text": "Variable \"ptr\" was declared here.",

"richText": "Variable `ptr` was declared here."

},

"physicalLocation": {

"fileLocation": {

"uri": "collections/list.h",

"uriBaseId": "SRCROOT",

"fileIndex": 3

},

"region": {

"startLine": 8,

"startColumn": 5

}

},

"fullyQualifiedLogicalName": "collections::list:add",

"logicalLocationIndex": 0

}

],

"codeFlows": [

{

"message": {

"text": "Path from declaration to usage"

},

"threadFlows": [

{

"id": "thread-52",

"locations": [

{

"importance": "essential",

"location": {

"message": {

"text": "Variable \"ptr\" declared.",

"richText": "Variable `ptr` declared."

},

"physicalLocation": {

"fileLocation": {

"uri":"collections/list.h",

"uriBaseId": "SRCROOT",

"fileIndex": 3

},

"region": {

"startLine": 15,

"snippet": {

"text": "int \*ptr;"

}

}

},

"fullyQualifiedLogicalName": "collections::list:add",

"logicalLocationIndex": 0

},

"module": "platform"

},

{

"state": {

"y": "2",

"z": "4",

"y + z": "6",

"q": "7"

},

"importance": "unimportant",

"location": {

"physicalLocation": {

"fileLocation": {

"uri":"collections/list.h",

"uriBaseId": "SRCROOT",

"fileIndex": 3

},

"region": {

"startLine": 15,

"snippet": {

"text": "offset = (y + z) \* q + 1;"

}

}

},

"annotations": [

{

"startLine": 15,

"startColumn": 13,

"endColumn": 19,

"message": {

"text": "(y + z) = 42",

"richText": "`(y + z) = 42`"

}

}

],

"fullyQualifiedLogicalName": "collections::list:add",

"logicalLocationIndex": 0

},

"module": "platform"

},

{

"importance": "essential",

"location": {

"message": {

"text": "Uninitialized variable \"ptr\" passed to

method \"add\_core\".",

"richText": "Uninitialized variable `ptr` passed to

method `add\_core`."

},

"physicalLocation": {

"fileLocation": {

"uri":"collections/list.h",

"uriBaseId": "SRCROOT",

"fileIndex": 3

},

"region": {

"startLine": 25,

"snippet": {

"text": "add\_core(ptr, offset, val)"

}

}

},

"fullyQualifiedLogicalName": "collections::list:add",

"logicalLocationIndex": 0

},

"module": "platform"

}

]

}

]

}

],

"stacks": [

{

"message": {

"text": "Call stack resulting from usage of uninitialized variable."

},

"frames": [

{

"location": {

"message": {

"text": "Exception thrown."

},

"physicalLocation": {

"fileLocation": {

"uri": "collections/list.h",

"uriBaseId": "SRCROOT",

"fileIndex": 3

},

"region": {

"startLine": 110,

"startColumn": 15

}

},

"fullyQualifiedLogicalName": "collections::list:add\_core",

"logicalLocationIndex": 0

},

"module": "platform",

"threadId": 52,

"address": 10092852,

"offset": 16,

"parameters": [ "null", "0", "14" ]

},

{

"location": {

"physicalLocation": {

"fileLocation": {

"uri": "collections/list.h",

"uriBaseId": "SRCROOT",

"fileIndex": 3

},

"region": {

"startLine": 43,

"startColumn": 15

}

},

"fullyQualifiedLogicalName": "collections::list:add",

"logicalLocationIndex": 0

},

"module": "platform",

"threadId": 52,

"address": 10092176,

"offset": 84,

"parameters": [ "14" ]

},

{

"location": {

"physicalLocation": {

"fileLocation": {

"uri": "application/main.cpp",

"uriBaseId": "SRCROOT",

"fileIndex": 1

},

"region": {

"startLine": 28,

"startColumn": 9

}

},

"fullyQualifiedLogicalName": "main",

"logicalLocationIndex": 4

},

"module": "application",

"threadId": 52,

"address": 10091200,

"offset": 156

}

]

}

],

"fixes": [

{

"description": {

"text": "Initialize the variable to null"

},

"fileChanges": [

{

"fileLocation": {

"uri": "collections/list.h",

"uriBaseId": "SRCROOT",

"fileIndex": 3

},

"replacements": [

{

"deletedRegion": {

"startLine": 42

},

"insertedContent": {

"text": "A different line\n"

}

}

]

}

]

}

],

"hostedViewerUri":

"https://www.example.com/viewer/3918d370-c636-40d8-bf23-8c176043a2df",

"workItemUris": [

"https://github.com/example/project/issues/42",

"https://github.com/example/project/issues/54"

],

"provenance": {

"firstDetectionTimeUtc": "2016-07-15T14:20:42Z",

"firstDetectionRunInstanceGuid": "8F62D8A0-C14F-4516-9959-1A663BA6FB99",

"lastDetectionTimeUtc": "2016-07-16T14:20:42Z",

"lastDetectionRunInstanceGuid": "BC650830-A9FE-44CB-8818-AD6C387279A0",

"invocationIndex": 0

}

}

],

"resources": {

"rules": [

{

"id": "C2001",

"deprecatedIds": [

"CA2000"

],

"defaultLevel": "error",

"defaultRank": 95,

"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). |
| 11 | 2018/03/28 | Laurence J Golding | Incorporate changes for GitHub issues [#75](https://github.com/oasis-tcs/sarif-spec/issues/75), [#80](https://github.com/oasis-tcs/sarif-spec/issues/80), [#86](https://github.com/oasis-tcs/sarif-spec/issues/86), [#95](https://github.com/oasis-tcs/sarif-spec/issues/95), [#96](https://github.com/oasis-tcs/sarif-spec/issues/96), and [#133](https://github.com/oasis-tcs/sarif-spec/issues/133). |
| 12 | 2018/04/18 | Laurence J. Golding | Incorporate changes for GitHub issues [#46](https://github.com/oasis-tcs/sarif-spec/issues/46), [#98](https://github.com/oasis-tcs/sarif-spec/issues/98), [#99](https://github.com/oasis-tcs/sarif-spec/issues/99), [#107](https://github.com/oasis-tcs/sarif-spec/issues/107), [#108](https://github.com/oasis-tcs/sarif-spec/issues/108), [#11](https://github.com/oasis-tcs/sarif-spec/issues/113)3, [#119](https://github.com/oasis-tcs/sarif-spec/issues/119), [#120](https://github.com/oasis-tcs/sarif-spec/issues/120), [#125](https://github.com/oasis-tcs/sarif-spec/issues/125), and [#130](https://github.com/oasis-tcs/sarif-spec/issues/130). |
| 13 | 2018/05/03 | Laurence J. Golding | Incorporate changes for GitHub issues [#122](https://github.com/oasis-tcs/sarif-spec/issues/122), [#126](https://github.com/oasis-tcs/sarif-spec/issues/126), [#134](https://github.com/oasis-tcs/sarif-spec/issues/134), [#136](https://github.com/oasis-tcs/sarif-spec/issues/136), [#137](https://github.com/oasis-tcs/sarif-spec/issues/137), [#139](https://github.com/oasis-tcs/sarif-spec/issues/139), [#145](https://github.com/oasis-tcs/sarif-spec/issues/145), [#147](https://github.com/oasis-tcs/sarif-spec/issues/147), [#154](https://github.com/oasis-tcs/sarif-spec/issues/154), and [#155](https://github.com/oasis-tcs/sarif-spec/issues/155).  Editorial change in result.ruleMessageId. |
| 14 | 2018/05/08 | Laurence J. Golding | Address GitHub issue [#156](https://github.com/oasis-tcs/sarif-spec/issues/156): editorial |
| 15 | 2018/05/17 | Laurence J. Golding | Incorporate changes for GitHub issues [#103](https://github.com/oasis-tcs/sarif-spec/issues/103), [#138](https://github.com/oasis-tcs/sarif-spec/issues/138), [#141](https://github.com/oasis-tcs/sarif-spec/issues/141), [#143](https://github.com/oasis-tcs/sarif-spec/issues/143), [#153](https://github.com/oasis-tcs/sarif-spec/issues/153), [#157](https://github.com/oasis-tcs/sarif-spec/issues/157), [#159](https://github.com/oasis-tcs/sarif-spec/issues/159), [#160](https://github.com/oasis-tcs/sarif-spec/issues/160), [#161](https://github.com/oasis-tcs/sarif-spec/issues/161), [#162](https://github.com/oasis-tcs/sarif-spec/issues/162), [#163](https://github.com/oasis-tcs/sarif-spec/issues/163), [#165](https://github.com/oasis-tcs/sarif-spec/issues/165), [#166](https://github.com/oasis-tcs/sarif-spec/issues/166), [#167](https://github.com/oasis-tcs/sarif-spec/issues/167), and [#170](https://github.com/oasis-tcs/sarif-spec/issues/170).  Editorial change for “occurs” *vs.* “contains”. |
| 16 | 2018/05/30 | Laurence J. Golding | Incorporate changes for GitHub issues [#93](https://github.com/oasis-tcs/sarif-spec/issues/93), [#149](https://github.com/oasis-tcs/sarif-spec/issues/149), [#160](https://github.com/oasis-tcs/sarif-spec/issues/160) (revised), [#171](https://github.com/oasis-tcs/sarif-spec/issues/171), [#176](https://github.com/oasis-tcs/sarif-spec/issues/176), [#181](https://github.com/oasis-tcs/sarif-spec/issues/181), and [#187](https://github.com/oasis-tcs/sarif-spec/issues/187) (editorial).  Editorial change: Remove “semanticVersion” from all but “Comprehensive” example in Appendix I.  Editorial change: Improve language for default values. |
| 17 | 2018/06/06 | Laurence J. Golding | Incorporate changes for GitHub issues [#158](https://github.com/oasis-tcs/sarif-spec/issues/158), [#164](https://github.com/oasis-tcs/sarif-spec/issues/164), [#172](https://github.com/oasis-tcs/sarif-spec/issues/172), [#175](https://github.com/oasis-tcs/sarif-spec/issues/175), [#178](https://github.com/oasis-tcs/sarif-spec/issues/178), and [#186](https://github.com/oasis-tcs/sarif-spec/issues/186). |
| 18 | 2018/06/08 | Laurence J. Golding | Incorporate changes for GitHub issues #189 and [#191](https://github.com/oasis-tcs/sarif-spec/issues/191). |
| 19 | 2018/11/14 | Laurence J. Golding | Incorporate changes for GitHub issues [#169](https://github.com/oasis-tcs/sarif-spec/issues/169), [#256](https://github.com/oasis-tcs/sarif-spec/issues/256), [#269](https://github.com/oasis-tcs/sarif-spec/issues/269), [#272](https://github.com/oasis-tcs/sarif-spec/issues/272), and [#275](https://github.com/oasis-tcs/sarif-spec/issues/275). |
| 20 | 2018/11/29 | Laurence J. Golding | Incorporate changes for GitHub issues [#186](https://github.com/oasis-tcs/sarif-spec/issues/186), [#188](https://github.com/oasis-tcs/sarif-spec/issues/188), [#274](https://github.com/oasis-tcs/sarif-spec/issues/274), [#279](https://github.com/oasis-tcs/sarif-spec/issues/279), [#280](https://github.com/oasis-tcs/sarif-spec/issues/280), [#284](https://github.com/oasis-tcs/sarif-spec/issues/284), [#285](https://github.com/oasis-tcs/sarif-spec/issues/285), and [#288](https://github.com/oasis-tcs/sarif-spec/issues/288). |
| 21 | 2018/12/13 | Laurence J. Golding | Incorporate changes for GitHub issues [#248](https://github.com/oasis-tcs/sarif-spec/issues/248), [#270](https://github.com/oasis-tcs/sarif-spec/issues/270), [#287](https://github.com/oasis-tcs/sarif-spec/issues/287), #[292](https://github.com/oasis-tcs/sarif-spec/issues/292), #[293](https://github.com/oasis-tcs/sarif-spec/issues/293), and #[297](https://github.com/oasis-tcs/sarif-spec/issues/297). |
| 22 | 2019/01/10 | Laurence J. Golding | Incorporate changes for GitHub issues [#286](https://github.com/oasis-tcs/sarif-spec/issues/286), [#291](https://github.com/oasis-tcs/sarif-spec/issues/291), [#303](https://github.com/oasis-tcs/sarif-spec/issues/303), and [#304](https://github.com/oasis-tcs/sarif-spec/issues/304). |
| 23 | 2019/02/20 | Laurence J. Golding | Incorporate changes for GitHub issues [#146](https://github.com/oasis-tcs/sarif-spec/issues/146), [#312](https://github.com/oasis-tcs/sarif-spec/issues/312), [#317](https://github.com/oasis-tcs/sarif-spec/issues/317), and [#322](https://github.com/oasis-tcs/sarif-spec/issues/322). |

1. Pronounced 'sæ-rɪf (“a” as in “cat”, “i” as in “if”, emphasis on the first syllable). [↑](#footnote-ref-1)