Digital Signature Service Core Protocols, Elements, and Bindings Version 2.0

Working Draft 01

DD Month YYYY

Technical Committee:

[OASIS Digital Signature Services eXtended (DSS-X) TC](https://www.oasis-open.org/committees/dss-x/)

Chairs:

Juan Carlos Cruellas ([cruellas@ac.upc.edu](mailto:cruellas@ac.upc.edu)), [Univ Politecnica de Cataluna](http://www.ac.upc.edu/)

Stefan Hagen ([stefan@hagen.link](mailto:stefan@hagen.link)), Individual

Editor:

Stefan Hagen ([stefan@hagen.link](mailto:stefan@hagen.link)), Individual

Additional artifacts:

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

* JSON and XML schemas: <http://docs.oasis-open.org/dss-x/dss-core/v2.0/csd01/schemas/>

Related work:

This specification replaces or supersedes:

* Stefan Drees et al., Digital Signature Service Core Protocols, Elements, and Bindings, Version 1.0, OASIS Standard, 11 April 2007,  
  <http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf>

This specification is related to:

* Related specifications (hyperlink, if available)

Declared XML namespaces:

* <http://docs.oasis-open.org/dss-x/ns/dss-core/v2.0/dss>

Abstract:

This document defines JSON and XML based request/response protocols for signing and verifying documents and other data. It also defines a timestamp format, and a signature property for use with these protocols. Finally, it defines transport and security bindings for the protocols.

Status:

This [Working Draft](https://www.oasis-open.org/policies-guidelines/tc-process) (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) 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) 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.

Any machine-readable content ([Computer Language Definitions](https://www.oasis-open.org/policies-guidelines/tc-process)) 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/dss-x/dss-core/v2.0/csd01/dss-core-v2.0-csd01.docx

Permanent “Latest version” URI:  
http://docs.oasis-open.org/dss-x/dss-core/v2.0/dss-core-v2.0.docx

(Managed by OASIS TC Administration; please don’t modify.)

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 9](#_Toc482893695)

[1.1 Organization of DSS Core Protocols, Elements, and Bindings 9](#_Toc482893696)

[1.2 Terminology 9](#_Toc482893697)

[1.2.1 Terms and Definitions 9](#_Toc482893698)

[1.2.2 Abbreviated Terms 9](#_Toc482893699)

[1.3 Normative References 9](#_Toc482893700)

[1.4 Non-Normative References 11](#_Toc482893701)

[1.5 Typographical Conventions 11](#_Toc482893702)

[2 Design Considerations 12](#_Toc482893703)

[2.1 Construction Principles 12](#_Toc482893704)

[2.2 Domain Models 12](#_Toc482893705)

[2.2.1 Date and Time Model 12](#_Toc482893706)

[2.3 Schema Organization and Namespaces 12](#_Toc482893707)

[2.4 DSS Overview (Non-normative) 13](#_Toc482893708)

[2.5 Version 2.0 motivation [non-normative] 14](#_Toc482893709)

[2.6 Syntax variants 14](#_Toc482893710)

[3 Structure Models 15](#_Toc482893711)

[3.1 Type Base64DataType 15](#_Toc482893712)

[3.2 Type AnyType 16](#_Toc482893713)

[3.3 Type InternationalStringType 16](#_Toc482893714)

[3.4 Type KeyInfoType 17](#_Toc482893715)

[3.5 Element InputDocuments 18](#_Toc482893716)

[3.5.1 Type DocumentBaseType 19](#_Toc482893717)

[3.5.2 Type DocumentType 19](#_Toc482893718)

[3.5.2.1 XML Syntax 19](#_Toc482893719)

[3.5.2.2 JSON Syntax 20](#_Toc482893720)

[3.5.3 Type TransformedDataType 20](#_Toc482893721)

[3.5.3.1 XML Syntax 21](#_Toc482893722)

[3.5.3.2 JSON Syntax 21](#_Toc482893723)

[3.5.4 Type DocumentHashType 21](#_Toc482893724)

[3.5.4.1 XML Syntax 22](#_Toc482893725)

[3.5.4.2 JSON Syntax 22](#_Toc482893726)

[3.6 Element SignatureObject 22](#_Toc482893727)

[3.6.1 XML Syntax 23](#_Toc482893728)

[3.6.2 JSON Syntax 24](#_Toc482893729)

[3.7 Element Result 24](#_Toc482893730)

[3.7.1 XML Syntax 26](#_Toc482893731)

[3.7.2 JSON Syntax 26](#_Toc482893732)

[3.8 Common Optional Inputs 27](#_Toc482893733)

[3.8.1 Optional Input ServicePolicy 27](#_Toc482893734)

[3.8.1.1 XML Syntax 27](#_Toc482893735)

[3.8.1.2 JSON Syntax 27](#_Toc482893736)

[3.8.2 Optional Input ClaimedIdentity 27](#_Toc482893737)

[3.8.2.1 XML Syntax 28](#_Toc482893738)

[3.8.2.2 JSON Syntax 28](#_Toc482893739)

[3.8.3 Optional Input Language 28](#_Toc482893740)

[3.8.3.1 XML Syntax 28](#_Toc482893741)

[3.8.3.2 JSON Syntax 28](#_Toc482893742)

[3.8.4 Optional Input Profile 29](#_Toc482893743)

[3.8.4.1 XML Syntax 29](#_Toc482893744)

[3.8.4.2 JSON Syntax 29](#_Toc482893745)

[3.8.5 Optional Input Schemas 29](#_Toc482893746)

[3.8.5.1 XML Syntax 30](#_Toc482893747)

[3.8.5.2 JSON Syntax 30](#_Toc482893748)

[3.8.6 Type Optional Input ReturnTransformedDocument and Output TransformedDocument 30](#_Toc482893749)

[3.8.6.1 XML Syntax 31](#_Toc482893750)

[3.8.6.2 JSON Syntax 31](#_Toc482893751)

[3.9 OptionalInputsBaseType 31](#_Toc482893752)

[3.9.1.1 XML Syntax 32](#_Toc482893753)

[3.9.1.2 JSON Syntax 32](#_Toc482893754)

[3.10 Common Optional Outputs 33](#_Toc482893755)

[3.10.1 Optional Output Schemas 33](#_Toc482893756)

[3.11 OptionalOutputsBaseType 33](#_Toc482893757)

[3.11.1.1 XML Syntax 33](#_Toc482893758)

[3.11.1.2 JSON Syntax 34](#_Toc482893759)

[3.12 Type RequestBaseType 34](#_Toc482893760)

[3.12.1 XML Syntax 35](#_Toc482893761)

[3.12.2 JSON Syntax 35](#_Toc482893762)

[3.13 Type ResponseBaseType 35](#_Toc482893763)

[3.13.1 XML Syntax 35](#_Toc482893764)

[3.13.2 JSON Syntax 35](#_Toc482893765)

[4 The DSS Signing Protocol 37](#_Toc482893766)

[4.1 Element SignRequest 37](#_Toc482893767)

[4.1.1 XML Syntax 37](#_Toc482893768)

[4.1.2 JSON Syntax 37](#_Toc482893769)

[4.2 Element SignResponse 37](#_Toc482893770)

[4.2.1 XML Syntax 38](#_Toc482893771)

[4.2.2 JSON Syntax 38](#_Toc482893772)

[4.3 Processing for XML Signatures 39](#_Toc482893773)

[4.3.1 Basic Process for XML 39](#_Toc482893774)

[4.3.2 Process Variant for TransformedData 40](#_Toc482893775)

[4.3.3 Process Variant for DocumentHash 40](#_Toc482893776)

[4.4 Basic Processing for CMS Signatures 41](#_Toc482893777)

[4.4.1 Process Variant for DocumentHash 41](#_Toc482893778)

[4.5 Optional Inputs and Outputs 41](#_Toc482893779)

[4.5.1 Optional Input SignatureType 42](#_Toc482893780)

[4.5.1.1 XML Syntax 42](#_Toc482893781)

[4.5.1.2 JSON Syntax 42](#_Toc482893782)

[4.5.2 Optional Input AddTimestamp 42](#_Toc482893783)

[4.5.2.1 XML Syntax 42](#_Toc482893784)

[4.5.2.2 JSON Syntax 42](#_Toc482893785)

[4.5.2.3 Processing of signatures time-stamping 43](#_Toc482893786)

[4.5.2.3.1 Processing for CMS signatures time-stamping 43](#_Toc482893787)

[4.5.2.3.2 Processing for XML Timestamps on XML signatures 43](#_Toc482893788)

[4.5.2.4 Processing for RFC 3161 Timestamps on XML signatures 44](#_Toc482893789)

[4.5.3 Optional Input IntendedAudience 44](#_Toc482893790)

[4.5.3.1 XML Syntax 44](#_Toc482893791)

[4.5.3.2 JSON Syntax 44](#_Toc482893792)

[4.5.4 Optional Input KeySelector 45](#_Toc482893793)

[4.5.4.1 XML Syntax 45](#_Toc482893794)

[4.5.4.2 JSON Syntax 45](#_Toc482893795)

[4.5.5 Optional Input Properties 45](#_Toc482893796)

[4.5.5.1 XML Syntax 45](#_Toc482893797)

[4.5.5.2 JSON Syntax 46](#_Toc482893798)

[4.5.6 Optional Input IncludeObject 46](#_Toc482893799)

[4.5.6.1 XML Syntax 47](#_Toc482893800)

[4.5.6.2 JSON Syntax 47](#_Toc482893801)

[4.5.6.3 XML Signatures Variant Optional Input IncludeObject 47](#_Toc482893802)

[4.5.7 Optional Input IncludeEContent 48](#_Toc482893803)

[4.5.8 Enveloped Signatures, Optional Input SignaturePlacement and Output DocumentWithSignature 49](#_Toc482893804)

[4.5.8.1 XML Syntax 50](#_Toc482893805)

[4.5.8.2 JSON Syntax 51](#_Toc482893806)

[4.5.9 Optional Input SignedReferences 51](#_Toc482893807)

[4.5.9.1 XML Syntax 53](#_Toc482893808)

[4.5.9.2 JSON Syntax 53](#_Toc482893809)

[4.6 OptionalInputsSignType 53](#_Toc482893810)

[4.6.1.1 XML Syntax 54](#_Toc482893811)

[4.6.1.2 JSON Syntax 54](#_Toc482893812)

[4.7 OptionalOutputsSignType 55](#_Toc482893813)

[4.7.1.1 XML Syntax 55](#_Toc482893814)

[4.7.1.2 JSON Syntax 55](#_Toc482893815)

[5 The DSS Verifying Protocol 57](#_Toc482893816)

[5.1 Element VerifyRequest 57](#_Toc482893817)

[5.1.1 XML Syntax 57](#_Toc482893818)

[5.1.2 JSON Syntax 57](#_Toc482893819)

[5.2 Element VerifyResponse 58](#_Toc482893820)

[5.2.1 JSON Syntax 58](#_Toc482893821)

[5.3 Basic Processing for XML Signatures 58](#_Toc482893822)

[5.3.1 Multi-Signature Verification 59](#_Toc482893823)

[5.3.2 Signature Timestamp verification procedure 60](#_Toc482893824)

[5.3.2.1 Processing for RFC 3161 Timestamp tokens on CMS Signatures. 60](#_Toc482893825)

[5.3.2.2 Processing for XML timestamp tokens on XML signatures 60](#_Toc482893826)

[5.3.2.3 Processing for RFC 3161 timestamp tokens on XML Signatures 61](#_Toc482893827)

[5.4 Basic Processing for CMS Signatures 62](#_Toc482893828)

[5.5 Optional Inputs and Outputs 62](#_Toc482893829)

[5.5.1 Optional Input VerifyManifests and Output VerifyManifestResults 62](#_Toc482893830)

[5.5.1.1 XML Syntax 62](#_Toc482893831)

[5.5.1.2 JSON Syntax 63](#_Toc482893832)

[5.5.2 Optional Input UseVerificationTime 63](#_Toc482893833)

[5.5.2.1 XML Syntax 63](#_Toc482893834)

[5.5.2.2 JSON Syntax 64](#_Toc482893835)

[5.5.3 Optional Input/Output ReturnVerificationTimeInfo / VerificationTimeInfo 64](#_Toc482893836)

[5.5.3.1 XML Syntax 65](#_Toc482893837)

[5.5.3.2 JSON Syntax 65](#_Toc482893838)

[5.5.4 Optional Input AdditionalKeyInfo 66](#_Toc482893839)

[5.5.4.1 XML Syntax 66](#_Toc482893840)

[5.5.4.2 JSON Syntax 66](#_Toc482893841)

[5.5.5 Optional Input ReturnProcessingDetails and Output ProcessingDetails 66](#_Toc482893842)

[5.5.5.1 XML Syntax 67](#_Toc482893843)

[5.5.5.2 JSON Syntax 68](#_Toc482893844)

[5.5.6 Optional Input ReturnSigningTimeInfo and Output SigningTimeInfo 68](#_Toc482893845)

[5.5.6.1 XML Syntax 69](#_Toc482893846)

[5.5.6.2 JSON Syntax 69](#_Toc482893847)

[5.5.7 Optional Input ReturnSignerIdentity and Output SignerIdentity 70](#_Toc482893848)

[5.5.7.1 XML Syntax 70](#_Toc482893849)

[5.5.7.2 JSON Syntax 70](#_Toc482893850)

[5.5.8 Optional Input ReturnUpdatedSignature and Outputs DocumentWithSignature, UpdatedSignature 70](#_Toc482893851)

[5.5.8.1 XML Syntax 71](#_Toc482893852)

[5.5.8.2 JSON Syntax 72](#_Toc482893853)

[5.5.9 Optional Input ReturnTransformedDocument and Output TransformedDocument 72](#_Toc482893854)

[5.5.9.1 XML Syntax 72](#_Toc482893855)

[5.5.9.2 JSON Syntax 73](#_Toc482893856)

[5.5.10 Optional Input ReturnTimestampedSignature and Outputs DocumentWithSignature, TimestampedSignature 73](#_Toc482893857)

[5.5.10.1 XML Syntax 74](#_Toc482893858)

[5.5.10.2 JSON Syntax 74](#_Toc482893859)

[5.6 OptionalInputsVerifyType 74](#_Toc482893860)

[5.6.1.1 XML Syntax 75](#_Toc482893861)

[5.6.1.2 JSON Syntax 75](#_Toc482893862)

[5.7 OptionalOutputsVerifyType 76](#_Toc482893863)

[5.7.1.1 XML Syntax 77](#_Toc482893864)

[5.7.1.2 JSON Syntax 77](#_Toc482893865)

[6 DSS Core Elements 78](#_Toc482893866)

[6.1 Element Timestamp 78](#_Toc482893867)

[6.1.1 XML Timestamp Token 78](#_Toc482893868)

[6.1.2 Element TstInfo 79](#_Toc482893869)

[6.1.2.1 XML Syntax 79](#_Toc482893870)

[6.1.2.2 JSON Syntax 79](#_Toc482893871)

[6.2 Element RequesterIdentity 80](#_Toc482893872)

[6.2.1.1 XML Syntax 80](#_Toc482893873)

[6.2.1.2 JSON Syntax 80](#_Toc482893874)

[7 DSS Core Bindings 81](#_Toc482893875)

[7.1 HTTP POST Transport Binding 81](#_Toc482893876)

[7.2 SOAP 1.2 Transport Binding 81](#_Toc482893877)

[7.2.1 SOAP Attachment Feature and Element <AttachmentReference> 82](#_Toc482893878)

[7.2.1.1 Signing Protocol, Processing for XML Signatures, Process Variant for <AttachmentReference> 82](#_Toc482893879)

[7.2.1.2 Verifying Protocol, Processing for XML Signatures, Process Variant for <AttachmentReference> 83](#_Toc482893880)

[7.2.1.3 Signing Protocol, Basic Processing for CMS Signatures, Process Variant for <AttachmentReference> 83](#_Toc482893881)

[7.2.1.4 Verifying Protocol, Basic Processing for CMS Signatures, Process Variant for <AttachmentReference> 83](#_Toc482893882)

[8 Processing Model 84](#_Toc482893883)

[9 JSON Format 85](#_Toc482893884)

[9.1 JSON – Type Base64DataType 85](#_Toc482893885)

[9.2 JSON – Type AnyType 85](#_Toc482893886)

[9.3 JSON – Type InternationalStringType 86](#_Toc482893887)

[9.4 JSON – Type KeyInfoType 86](#_Toc482893888)

[9.5 JSON – Element InputDocuments 87](#_Toc482893889)

[9.5.1 JSON – Type DocumentBaseType 87](#_Toc482893890)

[10 XML Format 89](#_Toc482893891)

[10.1 XML – Type Base64DataType 89](#_Toc482893892)

[10.2 XML – Type AnyType 89](#_Toc482893893)

[10.3 XML – Type InternationalStringType 90](#_Toc482893894)

[10.4 XML – Type KeyInfoType 90](#_Toc482893895)

[10.5 XML – Element InputDocuments 91](#_Toc482893896)

[10.5.1 XML – Type DocumentBaseType 92](#_Toc482893897)

[10.6 AnElement – REMOVE\_ME\_AFTER\_FIRST\_PASS 93](#_Toc482893898)

[11 DSS-Defined Identifiers 95](#_Toc482893899)

[11.1 Signature Type Identifiers 95](#_Toc482893900)

[11.1.1 XML Signature 95](#_Toc482893901)

[11.1.2 XML TimeStampToken 95](#_Toc482893902)

[11.1.3 RFC 3161 TimeStampToken 95](#_Toc482893903)

[11.1.4 CMS Signature 95](#_Toc482893904)

[11.1.5 PGP Signature 95](#_Toc482893905)

[12 Conformance 96](#_Toc482893906)

[12.1 Conformance as a DSS version 2.0 document 96](#_Toc482893907)

[12.1.1 Conformance for XML format 96](#_Toc482893908)

[12.1.2 Conformance for JSON format 96](#_Toc482893909)

[Appendix A. Acknowledgments 97](#_Toc482893910)

[Appendix B. 98](#_Toc482893911)

[B.1 Use of Exclusive Canonicalization 98](#_Toc482893912)

[B.2 More Complex Response Example 98](#_Toc482893913)

[Appendix C. 99](#_Toc482893914)

[C.1 Element InputDocuments 99](#_Toc482893915)

[C.1.1 XML Syntax 100](#_Toc482893916)

[C.1.2 JSON Syntax 100](#_Toc482893917)

[C.1.3 Type TransformedDataType 101](#_Toc482893918)

[Appendix D. Table of Types, Elements and Attributes 103](#_Toc482893919)

[Appendix E. List of Figures 105](#_Toc482893920)

[Appendix F. Index 106](#_Toc482893921)

[Appendix G. JSON Helpers 107](#_Toc482893922)

[Appendix H. Revision History 108](#_Toc482893923)

# Introduction

## Organization of DSS Core Protocols, Elements, and Bindings

The specification is split into twelve chapters.

## Terminology

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

### Terms and Definitions

For the purposes of this document, the following applies:

**Term** — meaning and maybe ref

### Abbreviated Terms

**Acronym** — Spelled out

## Normative References

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

**[RFC 2396]** T. Berners-Lee et al. *Uniform Resource Identifiers (URI): Generic Syntax.* IETF RFC 2396, August 1998.   
<http://www.ietf.org/rfc/rfc2396.txt>.

[DSS2XSD] S. Hagen,. *DSS 2.0 Schema*. OASIS, ToDo.

**[RFC 2440]** J. Callas, L. Donnerhacke, H. Finney, R. Thayer. *OpenPGP Message Format*. IETF RFC 2440, November 1998.  
<http://www.ietf.org/rfc/rfc2440.txt>.

**[RFC 2616]** R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616, June 1999.  
[http://www.ietf.org/rfc/rfc2616.txt](http://www.ietf.org/rfc/rfc2440.txt).

**[RFC 2648]** R. Moats. *A URN Namespace for IETF Documents*. IETF RFC 2648, August 1999.   
<http://www.ietf.org/rfc/rfc2648.txt>.

**[RFC 2822]** P. Resnick. *Internet Message Format*. IETF RFC 2822, April 2001. <http://www.ietf.org/rfc/rfc2822.txt>

**[RFC 3161]** C. Adams, P. Cain, D. Pinkas, R. Zuccherato. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. IETF RFC 3161, August 2001.   
[http://www.ietf.org/rfc/rfc3161.txt](http://www.ietf.org/rfc/rfc3075.txt).

**[RFC 5280]** D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. IETF RFC 5280, May 2008.   
<http://www.ietf.org/rfc/rfc5280.txt>.

**[RFC 5652]** R. Housley. *Cryptographic Message Syntax*. IETF RFC 5652, September 2009.  
<http://www.ietf.org/rfc/rfc5652.txt>.   
(Remark: As used in DSS, all implementations based upon RFC 5652 and previous releases of CMS will suffice. For the sake of simplicity the "urn:ietf:rfc:3369" is used throughout the document to indicate a CMS message as specified in RFC 5652 or RFC 3369 or any version (including PKCS #7).

**[****RFC7159]** T. Bray, Ed., Google, Inc., The JavaScript Object Notation (JSON) Data Interchange Format, ISSN: 2070-1721, March 2014.   
<https://tools.ietf.org/html/rfc7159>.

**[SAMLCore1.1]** E. Maler et al. Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V 1.1. OASIS, November 2002.   
<http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf>

**[SOAP]** M. Gudgin et al. *SOAP Version 1.2 Part 1: Messaging Framework.* W3C Recommendation, June 2003.   
<http://www.w3.org/TR/xmlschema-1/>

**[SOAPAtt]** H. F. Nielsen, H. Ruellan *SOAP 1.2 Attachment Feature,* W3C Working Group Note, 8 June 2004  
<http://www.w3.org/TR/soap12-af/>

**[WS-I-Att]** Ch. Ferris, A. Karmarkar, C. K. Liu *Attachments Profile Version 1.0,* The Web Services-Interoperability Organization (WS-I)*,* 20 April 2006  
<http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html>

**[XML-C14N]** J. Boyer. *Canonical XML Version 1.0*. W3C Recommendation, March 2001.   
<http://www.w3.org/TR/xml-c14n>

**[XML-xcl-c14n]** Exclusive XML Canonicalization Version 1.0. W3C Recommendation 18 July 2002 <http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/>

**[xml:id]** xml:id, Version 1.0, W3C Recommendation, 9 September 2005, http://www.w3.org/TR/xml-id/

**[XML-ns]** T. Bray, D. Hollander, A. Layman. *Namespaces in XML.* W3C Recommendation, January 1999.   
[http://www.w3.org/TR/1999/REC-xml-names-19990114](http://www.w3.org/TR/1999/REC-xml-names-19990114/)

**[XML-NT-Document]** [http://www.w3.org/TR/2004/REC-xml-20040204/#NT-document](http://www.w3.org/TR/2004/REC-xml-20040204/)

**[XML-PROLOG]** Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, et al. *Prolog and Document Type Declaration* in *Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C Recommendation, 04 February 2004, <http://www.w3.org/TR/REC-xml/#sec-prolog-dtd>

**[XMLDSIG]** *D. Eastlake et al. XML-Signature Syntax and Processing. W3C Recommendation, February 2002.*<http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/>

[XML] Extensible Markup Language (XML) 1.0 (Fifth Edition), T. Bray, J. Paoli, M. Sperberg-McQueen, E. Maler, F. Yergeau, Editors, W3C Recommendation, November 26, 2008, <http://www.w3.org/TR/2008/REC-xml-20081126/>.   
Latest version available at <http://www.w3.org/TR/xml>.

[XML-Schema-1] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures, S. Gao, M. Sperberg-McQueen, H. Thompson, N. Mendelsohn, D. Beech, M. Maloney, Editors, W3C Recommendation, April 5, 2012,   
<http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/>.   
Latest version available at <http://www.w3.org/TR/xmlschema11-1/>.

[XML-Schema-2] W3C XML Schema Definition Language (XSD) 1.1 Part 2: DatatypesW3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, D. Peterson, S. Gao, A. Malhotra, M. Sperberg-McQueen, H. Thompson, Paul V. Biron, Editors, W3C Recommendation, April 5, 2012,   
<http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/>.   
Latest version available at <http://www.w3.org/TR/xmlschema11-2/>.

**[XPATH]** XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999 <http://www.w3.org/TR/xpath>

## Non-Normative References

[ISO8601] Data elements and interchange formats — Information interchange — Representation of dates and times, International Standard, ISO 8601:2004(E), December 1, 2004, <https://www.iso.org/standard/40874.html>.

## Typographical Conventions

Keywords defined by this specification use this monospaced font.

Normative source code uses this paragraph style.

Text following the special symbol («) – an opening Guillemet (or French quotation mark) – within this specification identifies conformance statements. Every conformance statement is separated from the following text with the special end symbol (») – a closing Guillemet, and has been assigned a reference that follows that end symbol in the format [dSS-section#-local#].

Some sections of this specification are illustrated with non-normative examples.

Example 1: text describing an example uses this paragraph style

Non-normative examples use this paragraph style.

All examples in this document are non-normative and informative only.

Representation-specific text is indented and marked with vertical lines.

Representation-Specific Headline

Normative representation-specific text

All other text is normative unless otherwise labeled e.g. like:

Non-normative Comment:

This is a pure informative comment that may be present, because the information conveyed is deemed useful advice or common pitfalls learned from implementer or operator experience and often given including the rationale.

# Design Considerations

Blurb

## Construction Principles

## Domain Models

### Date and Time Model

The specific concept of date and time used in this document is defined in this section and noted in subsequent usage as**:**

DateTime

« All date time values inside a DSS document MUST adhere to the ISO 8601 [[ISO8601](#refISO8601)] basic or extended Format (as given there in section 4.3.2 “Complete representations” and with the addition of decimal fractions for seconds, similar to ibid. section 4.2.2.4 “Representations with decimal fraction” but with the full stop (.) being the preferred separator for DSS). » [DSS-2.2.1-1].

## Schema Organization and Namespaces

The structures described in this specification are contained in the schema file **[Core2.0-XSD]**. All schema listings in the current document are excerpts from the schema file. In the case of a disagreement between the schema file and this document, the schema file takes precedence.

This schema is associated with the following XML namespace:

urn:oasis:names:tc:dss:2.0:core:schema

If a future version of this specification is needed, it will use a different namespace.

Conventional XML namespace prefixes are used in the schema:

* The prefix dss2: stands for the DSS core namespace **[DSS2XSD]**.
* The prefix ds: stands for the W3C XML Signature namespace **[XMLDSIG]**.
* The prefix xs: stands for the W3C XML Schema namespace **[Schema1]**.
* The prefix saml: stands for the OASIS SAML Schema namespace **[SAMLCore1.1]**.

Applications MAY use different namespace prefixes, and MAY use whatever namespace defaulting/scoping conventions they desire, as long as they are compliant with the Namespaces in XML specification **[XML-ns]**.

The following schema fragment defines the XML namespaces and other header information for the DSS core schema:

<xs:schema xmlns:dss2="urn:oasis:names:tc:dss:2.0:core:schema"

xmlns:ds="http://www.w3.org/2000/09/xmldsig#"

xmlns:xs="http://www.w3.org/2001/XMLSchema"

xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"

targetNamespace="urn:oasis:names:tc:dss:2.0:core:schema"

elementFormDefault="qualified"

attributeFormDefault="unqualified">

<xs:annotation>

<xs:documentation xml:lang="en">This Schema defines the Digital Signature Service Core Protocols, Elements, and Bindings Committee Draft 1 for Public Review</xs:documentation>

</xs:annotation>

<xs:import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>

<xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion" schemaLocation="http://www.oasis-open.org/committees/download.php/3408/oasis-sstc-saml-schema-protocol-1.1.xsd"/>

<xs:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="http://www.w3.org/2001/xml.xsd"/>

## DSS Overview (Non-normative)

This specification describes two request/response protocols:

1. a signing protocol and
2. a verifying protocol.

Through these protocols a client can send documents (or document hashes) to a server and receive back a signature on the documents; or send documents (or document hashes) and a signature to a server, and receive back an answer on whether the signature verifies the documents.

The elements in which the protocols are formulated are provided in a format agnostic language and also in JSON and XML format. Provided are additional mappings from the generic to the specific entities.

These protocol operations could be useful in a variety of contexts – for example, they could allow clients to access a single corporate key for signing press releases, with centralized access control, auditing, and archiving of signature requests. They could also allow clients to create and verify signatures without needing complex client software and configuration.

The signing and verifying protocols are chiefly designed to support the creation and verification of XML signatures **[XMLDSIG]**, XML timestamps (see section 5.1), binary timestamps **[RFC 3161]** and CMS signatures **[RFC 3852]**. These protocols may also be extensible to other types of signatures and timestamps, such as PGP signatures **[RFC 2440]**.

It is expected that the signing and verifying protocols will be *profiled* to meet many different application scenarios. In anticipation of this, these protocols have only a minimal set of required elements, which deal with transferring “input documents” and signatures back and forth between client and server. The input documents to be signed or verified can be transferred in their entirety or the client can hash the documents themselves and only send the hash values to save bandwidth and protect the confidentiality of the document content.

All functionality besides transferring input documents and signatures is relegated to a framework of “optional inputs” and “optional outputs”. This document defines a number of optional inputs and outputs. Profiles of these protocols can pick and choose which optional inputs and outputs to support and can introduce their own optional inputs and outputs when they need functionality not anticipated by this specification.

Examples of optional inputs to the signing protocol include: what type of signature to produce, which key to sign with, who the signature is intended for, and what signed and unsigned properties to place in the signature. Examples of optional inputs to the verifying protocol include: the time for which the client would like to know the signature’s validity status, additional validation data necessary to verify the signature (such as certificates and CRLs), and requests for the server to return information such as the signer’s name or the signing time.

The signing and verifying protocol messages must be transferred over some underlying protocol(s) which provide message transport and security. A *binding* specifies how to use the signing and verifying protocols with some underlying protocol such as HTTP POST or TLS. Section 6 provides an initial set of bindings.

In addition to defining the signing and verifying protocols, this specification defines two elements that are related to these protocols. First, an XML timestamp element is defined in section 5.1. The signing and verifying protocols can be used to create and verify both XML and binary timestamps; a profile for doing so is defined in **[XML-TSP]**. Second, a RequesterIdentity element is defined in section 5.2. This element can be used as a signature property in an XML signature, to give the name of the end-user who requested the signature.

## Version 2.0 motivation [non-normative]

The main changes of this version of the DSS/X core document compared to version 1.0 are:

1. include requirements that became known only after publication of version 1.0
2. simplify the core schema, e.g. by dropping elements seldom used.
3. support other transport formats than SOAP.

To guide the implementation and to ease the use of the protocol with common frameworks the following list of requirements was compiled:

* Focus on Base64 as the most versatile way to transport documents and signatures
* Avoid the use of XML specifics (like e.g. mixed content)
* Avoid xs:any by replacing it with an enumeration of possible types, and if that is not feasible, use base64 blobs as a fallback.
* Define cardinality of OptionalInputs and OptionalOutputs child elements explicitly
* Rearrange sequences and choices to produce a strongly typed object model
* The set of comments and bug reports arrived since version DSS 1.0 became standard were respected.

## Syntax variants

This version of the DSS/X core document handles the representation of requests and response elements according to the JSON and XML syntax. The general semantics of the elements is discussed in the element’s main section. Details of the JSON or XML formats are discussed in specific subsections

* JSON syntax
* XML syntax

# Structure Models

To specify data models, a distinction is made between a (data) type and a (data) component.

* A type specifies the meaning and structure of some information.
  + A type has a name.
  + A type can consist of a combination of data components and is referred to as a composite data type.
  + Base data types are: strings, integers and floats.
  + « If not stated otherwise, the base type string MUST be assumed as a base representation of the type. » [DSS-3-1]
* A component specifies an instance (a value) of a type.
  + A component has a name.
  + One can refer to the value of the component by using the name of the component.
  + « If the type of a component is not mentioned, a corresponding type MUST be assumed by appending the postfix “Type” to the name of the container. » [DSS-3-2]

## Type ContainerType

Type ContainerType

The ContainerType is a composite data type that can contain an arbitrary number of components. It is intended to provide arbitrary artifacts regardless of underlying transport format restrictions. This type is used to allow the transport of data not known at the design time of the schema.

« ContainerType MUST contain the elements AttRefUri, Id, IdRef, and MimeType all with the respective cardinalities [0, 1]. » [DSS-3.1-1]

« Only one of Id or IdRef children MUST be present within one Instance of ContainerType. » [DSS-3.1-2]

Element AttRefUri

The AttRefUri element holds any URI that references an attachment according to attachment transport specifications, e.g. **[SOAPAtt].**

Element Idannot assume a bilateral agreement between

The Id element defines a unique identification for this content element.

Element IdRef

The IdRef element references to another instance of the ContainerType with the ID element holding the same unique identifier value.

Element MimeType

The MimeType element holds a string, that describes the content type of the encoded data.

Non-normative Comment:

The (Id, IdRef) pair allows to include base64 encoded artifact just once even when it is used in different place in the protocol.

Typical use case are where the payload may contain binary parts that cannot be represented in the used transport format directly (e.g. null bytes), or it represents content that could interfere with the transport syntax, e.g. a XML document within a XML transport envelope.

Format specific implementations including mappings to the generic terms are defined in 9.1 [JSON – Type Base64DataType](#_JSON_–_Type) and 10.1 [XML – Type Base64DataType](#_XML_–_Type) respectively.

## Element AnyType

**Semantics**

AnyType holds generic content AnyType

Below follows a list of the sub-components that MAY be present within this component:

* The Content element MUST contain 1 or more sub\_components .
* The Base64Content element MUST hold base64 encoded binary data.
* The optional MimeType element MUST hold a string.

### XML Syntax

The XML element SHALL implement in XML syntax the AnyType component.

The AnyType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="AnyType">

<xs:sequence>

<xs:element name="Content" minOccurs="1" maxOccurs="unbounded">

<xs:complexType>

<xs:sequence>

<xs:element name="Base64Content" type="xs:base64Binary"/>

<xs:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>

<xs:attribute name="MimeType" type="xs:string" use="optional"/>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

Each child element of AnyType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The AnyType XML element SHALL NOT be empty.

### JSON Syntax

The AnyType JSON object SHALL implement in JSON syntax the AnyType component.

The AnyType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-AnyType": {

"type": "object"

"properties": {

"content": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-AnyType:Content"

}

}

}

}

"dss-AnyType:Content": {

"type": "object"

"properties": {

"b64Content": {

"type": "string"

}

"mimeType": {

"type": "string"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| Content | content |
| Base64Content | b64Content |
| MimeType | mimeType |

## Element InternationalStringType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The value element MUST hold a string.

### XML Syntax

The XML element SHALL implement in XML syntax the InternationalStringType component.

The InternationalStringType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="InternationalStringType">

<xs:simpleContent>

<xs:extension base="xs:string">

<xs:attribute ref="xml:lang" use="required"/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

Each child element of InternationalStringType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The InternationalStringType XML element SHALL NOT be empty.

### JSON Syntax

The InternationalStringType JSON object SHALL implement in JSON syntax the InternationalStringType component.

The InternationalStringType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-InternationalStringType": {

"type": "object"

"properties": {

"value": {

"type": "string"

}

"lang": {

"type": "string"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| value | value |
| lang | lang |

## Element DocumentBaseType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional ID element MUST hold a unique identifier.
* The optional RefURI element MUST hold an URI.
* The optional RefType element MUST hold an URI.
* The optional SchemaRefs element MUST hold a unique identifier reference.

### XML Syntax

The XML element SHALL implement in XML syntax the DocumentBaseType component.

The DocumentBaseType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="DocumentBaseType" abstract="true">

<xs:attribute name="ID" type="xs:ID" use="optional"/>

<xs:attribute name="RefURI" type="xs:anyURI" use="optional"/>

<xs:attribute name="RefType" type="xs:anyURI" use="optional"/>

<xs:attribute name="SchemaRefs" type="xs:IDREFS" use="optional"/>

</xs:complexType>

Each child element of DocumentBaseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The DocumentBaseType XML element SHALL NOT be empty.

### JSON Syntax

The DocumentBaseType JSON object SHALL implement in JSON syntax the DocumentBaseType component.

The DocumentBaseType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-DocumentBaseType": {

"type": "object"

"properties": {

"ID": {

"type": "string"

}

"refURI": {

"type": "string"

}

"refType": {

"type": "string"

}

"schemaRefs": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-DocumentType"

}

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| ID | ID |
| RefURI | refURI |
| RefType | refType |
| SchemaRefs | schemaRefs |

## Element DocumentType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional ID element MUST hold a unique identifier.
* The optional RefURI element MUST hold an URI.
* The optional RefType element MUST hold an URI.
* The optional SchemaRefs element MUST hold a unique identifier reference.
* The Base64Data element MUST hold a sub-component Base64DataType.

### XML Syntax

The XML element SHALL implement in XML syntax the DocumentType component.

The DocumentType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="DocumentType">

<xs:complexContent>

<xs:extension base="dss:DocumentBaseType">

<xs:choice>

<xs:element name="Base64Data" type="dss:Base64DataType"/>

</xs:choice>

</xs:extension>

</xs:complexContent>

</xs:complexType>

Each child element of DocumentType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The DocumentType XML element SHALL NOT be empty.

### JSON Syntax

The DocumentType JSON object SHALL implement in JSON syntax the DocumentType component.

The DocumentType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-DocumentType": {

"type": "object"

"properties": {

"ID": {

"type": "string"

}

"refURI": {

"type": "string"

}

"refType": {

"type": "string"

}

"schemaRefs": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-DocumentType"

}

}

"b64Data": {

"type": "object"

"$ref": "#/definitions/dss-Base64DataType"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| ID | ID |
| RefURI | refURI |
| RefType | refType |
| SchemaRefs | schemaRefs |
| Base64Data | b64Data |

## Element Base64DataType

**Semantics**

Base64DataType does some relevant stuff! This is the best Base64DataType you can get!

Below follows a list of the sub-components that MAY be present within this component:

* The value element MUST hold base64 encoded binary data.
* The optional MimeType element MUST hold a string. The mime type of the content.
* The optional AttRefURI element MUST hold an URI. The attachment reference URI.
* The optional ID element MUST hold a unique identifier. Identifier of this blob.
* The optional IDREF element MUST hold a unique identifier reference. Identifier of the referenced blob.

Non-normative Comment:

Non-normative comment

### XML Syntax

The XML element SHALL implement in XML syntax the Base64DataType component.

The Base64DataType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="Base64DataType">

<xs:simpleContent>

<xs:extension base="xs:base64Binary">

<xs:attribute name="MimeType" type="xs:string" use="optional"/>

<xs:attribute name="AttRefURI" type="xs:anyURI" use="optional"/>

<xs:attribute name="ID" type="xs:ID" use="optional"/>

<xs:attribute name="IDREF" type="xs:IDREF" use="optional"/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

Each child element of Base64DataType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The Base64DataType XML element SHALL NOT be empty.

### JSON Syntax

The Base64DataType JSON object SHALL implement in JSON syntax the Base64DataType component.

The Base64DataType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-Base64DataType": {

"type": "object"

"properties": {

"ID": {

"type": "string"

}

"value": {

"type": "string"

}

"mimeType": {

"type": "string"

}

"attRef": {

"type": "string"

}

"IDREF": {

"type": "string"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| value | value |
| MimeType | mimeType |
| AttRefURI | attRef |
| ID | ID |
| IDREF | IDREF |

## Element TransformedDataType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional ID element MUST hold a unique identifier.
* The optional RefURI element MUST hold an URI.
* The optional RefType element MUST hold an URI.
* The optional SchemaRefs element MUST hold a unique identifier reference.
* The Base64Data element MUST hold a sub-component Base64DataType.
* The Base64Data element MUST hold a sub-component Base64DataType.
* The optional WhichReference element MUST hold an integer.

### XML Syntax

The XML element SHALL implement in XML syntax the TransformedDataType component.

The TransformedDataType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="TransformedDataType">

<xs:complexContent>

<xs:extension base="dss:DocumentBaseType">

<xs:sequence>

<xs:element ref="ds:Transforms" minOccurs="0"/>

<xs:element ref="dss:Base64Data"/>

</xs:sequence>

<xs:attribute name="WhichReference" type="xs:integer" use="optional"/>

</xs:extension>

</xs:complexContent>

</xs:complexType>

Each child element of TransformedDataType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The TransformedDataType XML element SHALL NOT be empty.

### JSON Syntax

The TransformedDataType JSON object SHALL implement in JSON syntax the TransformedDataType component.

The TransformedDataType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-TransformedDataType": {

"type": "object"

"properties": {

"ID": {

"type": "string"

}

"refURI": {

"type": "string"

}

"refType": {

"type": "string"

}

"schemaRefs": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-DocumentType"

}

}

"transforms": {

"type": "object"

"$ref": "#/definitions/dsig-TransformsType"

}

"b64Data": {

"type": "object"

"$ref": "#/definitions/dss-Base64DataType"

}

"whichRef": {

"type": "integer"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| ID | ID |
| RefURI | refURI |
| RefType | refType |
| SchemaRefs | schemaRefs |
| Transforms | transforms |
| Base64Data | b64Data |
| WhichReference | whichRef |

## Element DocumentHashType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional ID element MUST hold a unique identifier.
* The optional RefURI element MUST hold an URI.
* The optional RefType element MUST hold an URI.
* The optional SchemaRefs element MUST hold a unique identifier reference.
* The DigestInfos element MUST contain 1 or more sub-components . Each one MUST satisfy the requirements specified in section DigestInfoType.
* The optional WhichReference element MUST hold an integer.

### XML Syntax

The XML element SHALL implement in XML syntax the DocumentHashType component.

The DocumentHashType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="DocumentHashType">

<xs:complexContent>

<xs:extension base="dss:DocumentBaseType">

<xs:sequence>

<xs:element ref="ds:Transforms" minOccurs="0"/>

<xs:element name="DigestInfos" type="dss:DigestInfoType" minOccurs="1" maxOccurs="unbounded"/>

</xs:sequence>

<xs:attribute name="WhichReference" type="xs:integer" use="optional"/>

</xs:extension>

</xs:complexContent>

</xs:complexType>

Each child element of DocumentHashType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The DocumentHashType XML element SHALL NOT be empty.

### JSON Syntax

The DocumentHashType JSON object SHALL implement in JSON syntax the DocumentHashType component.

The DocumentHashType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-DocumentHashType": {

"type": "object"

"properties": {

"ID": {

"type": "string"

}

"refURI": {

"type": "string"

}

"refType": {

"type": "string"

}

"schemaRefs": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-DocumentType"

}

}

"transforms": {

"type": "object"

"$ref": "#/definitions/dsig-TransformsType"

}

"di": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-DigestInfoType"

}

}

"whichRef": {

"type": "integer"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| ID | ID |
| RefURI | refURI |
| RefType | refType |
| SchemaRefs | schemaRefs |
| Transforms | transforms |
| DigestInfos | di |
| WhichReference | whichRef |

## Element DigestInfoType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The DigestMethod element MUST hold a string.
* The DigestValue element MUST hold base64 encoded binary data.

### XML Syntax

The XML element SHALL implement in XML syntax the DigestInfoType component.

The DigestInfoType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="DigestInfoType">

<xs:sequence>

<xs:element name="DigestMethod" type="xs:string"/>

<xs:element name="DigestValue" type="xs:base64Binary"/>

</xs:sequence>

</xs:complexType>

Each child element of DigestInfoType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The DigestInfoType XML element SHALL NOT be empty.

### JSON Syntax

The DigestInfoType JSON object SHALL implement in JSON syntax the DigestInfoType component.

The DigestInfoType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-DigestInfoType": {

"type": "object"

"properties": {

"alg": {

"type": "string"

}

"value": {

"type": "string"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| DigestMethod | alg |
| DigestValue | value |

## Element OptionalInputsBaseType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional Profile element MUST contain zero or more URIs.
* The ServicePolicy element MUST hold an URI.
* The ClaimedIdentity element MUST hold a sub\_component .
* The Language element MUST hold a language descriptor.
* The Schemas element MUST hold a sub-component SchemasType.
* The AddTimestamp element MUST hold a sub-component UpdateSignatureInstructionType.
* The optional Other element MUST contain zero or more sub-components . Each one MUST satisfy the requirements specified in section PropertyType.
* The optional SignatureForm element MUST contain an URI if present.

### XML Syntax

The XML element SHALL implement in XML syntax the OptionalInputsBaseType component.

The OptionalInputsBaseType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="OptionalInputsBaseType">

<xs:sequence>

<xs:element name="Profile" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>

<xs:element ref="dss:ServicePolicy" minOccurs="0" maxOccurs="unbounded"/>

<xs:element ref="dss:ClaimedIdentity" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:Language" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:Schemas" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:AddTimestamp" minOccurs="0" maxOccurs="1"/>

<xs:element name="Other" type="dss:PropertyType" minOccurs="0" maxOccurs="unbounded"/>

<xs:element name="SignatureForm" type="xs:anyURI" minOccurs="0" maxOccurs="1"/>

</xs:sequence>

</xs:complexType>

Each child element of OptionalInputsBaseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The OptionalInputsBaseType XML element SHALL NOT be empty.

### JSON Syntax

The OptionalInputsBaseType component has JSON schema represenation as it is not used directly.

## Element OptionalInputsSignType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional Profile element MUST contain zero or more URIs.
* The ServicePolicy element MUST hold an URI.
* The ClaimedIdentity element MUST hold a sub\_component .
* The Language element MUST hold a language descriptor.
* The Schemas element MUST hold a sub-component SchemasType.
* The AddTimestamp element MUST hold a sub-component UpdateSignatureInstructionType.
* The optional Other element MUST contain zero or more sub-components . Each one MUST satisfy the requirements specified in section PropertyType.
* The optional SignatureForm element MUST contain an URI if present.
* The SignatureType element MUST hold an URI.
* The IntendedAudience element MUST hold a sub\_component .
* The KeySelector element MUST hold a sub-component KeyInfoType.
* The Properties element MUST hold a sub\_component .
* The IncludeObject element MUST hold a sub\_component .
* The SignaturePlacement element MUST hold a sub\_component .
* The SignedReferences element MUST hold a sub\_component .
* The optional Nonce element MUST contain an integer if present.
* The optional SignatureAlgorithm element MUST contain an URI if present.
* The optional SignatureActivationData element MUST contain a string if present.

### XML Syntax

The XML element SHALL implement in XML syntax the OptionalInputsSignType component.

The OptionalInputsSignType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="OptionalInputsSignType">

<xs:complexContent>

<xs:extension base="dss:OptionalInputsBaseType">

<xs:sequence>

<xs:element ref="dss:SignatureType" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:IntendedAudience" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:KeySelector" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:Properties" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:IncludeObject" minOccurs="0" maxOccurs="unbounded"/>

<xs:element ref="dss:SignaturePlacement" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:SignedReferences" minOccurs="0" maxOccurs="1"/>

<xs:element name="Nonce" type="xs:integer" minOccurs="0" maxOccurs="1"/>

<xs:element name="SignatureAlgorithm" type="xs:anyURI" minOccurs="0" maxOccurs="1"/>

<xs:element name="SignatureActivationData" type="xs:string" minOccurs="0" maxOccurs="1"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

Each child element of OptionalInputsSignType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The OptionalInputsSignType XML element SHALL NOT be empty.

### JSON Syntax

The OptionalInputsSignType JSON object SHALL implement in JSON syntax the OptionalInputsSignType component.

The OptionalInputsSignType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-OptionalInputsSignType": {

"type": "object"

"properties": {

"profile": {

"type": "array"

"items": {

"type": "string"

}

}

"policy": {

"type": "array"

"items": {

"type": "string"

}

}

"claimedIdentity": {

"type": "object"

"$ref": "#/definitions/dss-ClaimedIdentity"

}

"lang": {

"type": "string"

}

"schemas": {

"type": "object"

"$ref": "#/definitions/dss-SchemasType"

}

"addTimestamp": {

"type": "object"

"$ref": "#/definitions/dss-UpdateSignatureInstructionType"

}

"other": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-PropertyType"

}

}

"sigForm": {

"type": "string"

}

"sigType": {

"type": "string"

}

"audience": {

"type": "object"

"$ref": "#/definitions/dss-IntendedAudience"

}

"keySel": {

"type": "object"

"$ref": "#/definitions/dss-KeyInfoType"

}

"props": {

"type": "object"

"$ref": "#/definitions/dss-Properties"

}

"includeObj": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-IncludeObject"

}

}

"sigPlacement": {

"type": "object"

"$ref": "#/definitions/dss-SignaturePlacement"

}

"sigRefs": {

"type": "object"

"$ref": "#/definitions/dss-SignedReferences"

}

"nonce": {

"type": "integer"

}

"sigAlgo": {

"type": "string"

}

"sad": {

"type": "string"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| Profile | profile |
| ServicePolicy | policy |
| ClaimedIdentity | claimedIdentity |
| Language | lang |
| Schemas | schemas |
| AddTimestamp | addTimestamp |
| Other | other |
| SignatureForm | sigForm |
| SignatureType | sigType |
| IntendedAudience | audience |
| KeySelector | keySel |
| Properties | props |
| IncludeObject | includeObj |
| SignaturePlacement | sigPlacement |
| SignedReferences | sigRefs |
| Nonce | nonce |
| SignatureAlgorithm | sigAlgo |
| SignatureActivationData | sad |

## Element OptionalInputsVerifyType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional Profile element MUST contain zero or more URIs.
* The ServicePolicy element MUST hold an URI.
* The ClaimedIdentity element MUST hold a sub\_component .
* The Language element MUST hold a language descriptor.
* The Schemas element MUST hold a sub-component SchemasType.
* The AddTimestamp element MUST hold a sub-component UpdateSignatureInstructionType.
* The optional Other element MUST contain zero or more sub-components . Each one MUST satisfy the requirements specified in section PropertyType.
* The optional SignatureForm element MUST contain an URI if present.
* The optional EvidenceRecords element MUST contain zero or more sub-components . Each one MUST satisfy the requirements specified in section AnyType.
* The optional POE element MUST contain zero or more sub-components . Each one MUST satisfy the requirements specified in section AnyType.
* The optional AdditionalConstraint element MUST contain zero or more URIs.
* The optional VerificationProcess element MUST contain a sub\_component if present. If present this MUST satisfy the requirements specified in section .
* The UseVerificationTime element MUST hold a sub-component UseVerificationTimeType.
* The ReturnVerificationTimeInfo element MUST hold a boolean.
* The AdditionalKeyInfo element MUST hold a sub\_component .
* The ReturnProcessingDetails element MUST hold a boolean.
* The ReturnSigningTimeInfo element MUST hold a boolean.
* The ReturnSignerIdentity element MUST hold a boolean.
* The ReturnUpdatedSignature element MUST hold a boolean.
* The ReturnTransformedDocument element MUST hold a sub\_component .
* The ReturnTimestampedSignature element MUST hold a boolean.

### XML Syntax

The XML element SHALL implement in XML syntax the OptionalInputsVerifyType component.

The OptionalInputsVerifyType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="OptionalInputsVerifyType">

<xs:complexContent>

<xs:extension base="dss:OptionalInputsBaseType">

<xs:sequence>

<xs:element ref="vr:ReturnVerificationReport" minOccurs="0" maxOccurs="1"/>

<xs:element name="EvidenceRecords" type="dss:AnyType" minOccurs="0" maxOccurs="unbounded"/>

<xs:element name="POE" type="dss:AnyType" minOccurs="0" maxOccurs="unbounded"/>

<xs:element name="AdditionalConstraint" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>

<xs:element name="VerificationProcess" type="vr:ValidationProcessType" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:UseVerificationTime" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:ReturnVerificationTimeInfo" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:AdditionalKeyInfo" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:ReturnProcessingDetails" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:ReturnSigningTimeInfo" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:ReturnSignerIdentity" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:ReturnUpdatedSignature" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:ReturnTransformedDocument" minOccurs="0" maxOccurs="unbounded"/>

<xs:element ref="dss:ReturnTimestampedSignature" minOccurs="0" maxOccurs="1"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

Each child element of OptionalInputsVerifyType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The OptionalInputsVerifyType XML element SHALL NOT be empty.

### JSON Syntax

The OptionalInputsVerifyType JSON object SHALL implement in JSON syntax the OptionalInputsVerifyType component.

The OptionalInputsVerifyType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-OptionalInputsVerifyType": {

"type": "object"

"properties": {

"profile": {

"type": "array"

"items": {

"type": "string"

}

}

"policy": {

"type": "array"

"items": {

"type": "string"

}

}

"claimedIdentity": {

"type": "object"

"$ref": "#/definitions/dss-ClaimedIdentity"

}

"lang": {

"type": "string"

}

"schemas": {

"type": "object"

"$ref": "#/definitions/dss-SchemasType"

}

"addTimestamp": {

"type": "object"

"$ref": "#/definitions/dss-UpdateSignatureInstructionType"

}

"other": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-PropertyType"

}

}

"sigForm": {

"type": "string"

}

"returnVerificationReport": {

"type": "object"

"$ref": "#/definitions/vr-ReturnVerificationReport"

}

"evidenceRecords": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-AnyType"

}

}

"POE": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-AnyType"

}

}

"addConstraint": {

"type": "array"

"items": {

"type": "string"

}

}

"verificationProcess": {

"type": "string"

}

"useVerificationTime": {

"type": "object"

"$ref": "#/definitions/dss-UseVerificationTimeType"

}

"returnVerificationTime": {

"type": "boolean"

}

"addKeyInfo": {

"type": "object"

"$ref": "#/definitions/dss-AdditionalKeyInfo"

}

"returnProcDetails": {

"type": "boolean"

}

"returnSigningTime": {

"type": "boolean"

}

"returnSigner": {

"type": "boolean"

}

"returnUpdated": {

"type": "boolean"

}

"returnTransformed": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-ReturnTransformedDocument"

}

}

"returnTimestamped": {

"type": "boolean"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| Profile | profile |
| ServicePolicy | policy |
| ClaimedIdentity | claimedIdentity |
| Language | lang |
| Schemas | schemas |
| AddTimestamp | addTimestamp |
| Other | other |
| SignatureForm | sigForm |
| ReturnVerificationReport | returnVerificationReport |
| EvidenceRecords | evidenceRecords |
| POE | POE |
| AdditionalConstraint | addConstraint |
| VerificationProcess | verificationProcess |
| UseVerificationTime | useVerificationTime |
| ReturnVerificationTimeInfo | returnVerificationTime |
| AdditionalKeyInfo | addKeyInfo |
| ReturnProcessingDetails | returnProcDetails |
| ReturnSigningTimeInfo | returnSigningTime |
| ReturnSignerIdentity | returnSigner |
| ReturnUpdatedSignature | returnUpdated |
| ReturnTransformedDocument | returnTransformed |
| ReturnTimestampedSignature | returnTimestamped |

## Element OptionalOutputsBaseType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional AppliedProfile element MUST contain zero or more URIs.
* The optional AppliedPolicy element MUST contain zero or more URIs.
* The TransformedDocument element MUST hold a sub\_component .
* The Schemas element MUST hold a sub-component SchemasType.
* The DocumentWithSignature element MUST hold a sub\_component .
* The optional Other element MUST contain zero or more sub-components . Each one MUST satisfy the requirements specified in section PropertyType.

### XML Syntax

The XML element SHALL implement in XML syntax the OptionalOutputsBaseType component.

The OptionalOutputsBaseType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="OptionalOutputsBaseType">

<xs:sequence>

<xs:element name="AppliedProfile" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>

<xs:element name="AppliedPolicy" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>

<xs:element ref="dss:TransformedDocument" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:Schemas" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:DocumentWithSignature" minOccurs="0" maxOccurs="1"/>

<xs:element name="Other" type="dss:PropertyType" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

Each child element of OptionalOutputsBaseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The OptionalOutputsBaseType XML element SHALL NOT be empty.

### JSON Syntax

The OptionalOutputsBaseType component has JSON schema represenation as it is not used directly.

## Element OptionalOutputsSignType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional AppliedProfile element MUST contain zero or more URIs.
* The optional AppliedPolicy element MUST contain zero or more URIs.
* The TransformedDocument element MUST hold a sub\_component .
* The Schemas element MUST hold a sub-component SchemasType.
* The DocumentWithSignature element MUST hold a sub\_component .
* The optional Other element MUST contain zero or more sub-components . Each one MUST satisfy the requirements specified in section PropertyType.

### XML Syntax

The XML element SHALL implement in XML syntax the OptionalOutputsSignType component.

The OptionalOutputsSignType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="OptionalOutputsSignType">

<xs:complexContent>

<xs:extension base="dss:OptionalOutputsBaseType">

<xs:sequence/>

</xs:extension>

</xs:complexContent>

</xs:complexType>

Each child element of OptionalOutputsSignType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The OptionalOutputsSignType XML element SHALL NOT be empty.

### JSON Syntax

The OptionalOutputsSignType JSON object SHALL implement in JSON syntax the OptionalOutputsSignType component.

The OptionalOutputsSignType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-OptionalOutputsSignType": {

"type": "object"

"properties": {

"profile": {

"type": "array"

"items": {

"type": "string"

}

}

"policy": {

"type": "array"

"items": {

"type": "string"

}

}

"transformed": {

"type": "object"

"$ref": "#/definitions/dss-TransformedDocument"

}

"schemas": {

"type": "object"

"$ref": "#/definitions/dss-SchemasType"

}

"docWithSignature": {

"type": "object"

"$ref": "#/definitions/dss-DocumentWithSignature"

}

"other": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-PropertyType"

}

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| AppliedProfile | profile |
| AppliedPolicy | policy |
| TransformedDocument | transformed |
| Schemas | schemas |
| DocumentWithSignature | docWithSignature |
| Other | other |

## Element OptionalOutputsVerifyType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional AppliedProfile element MUST contain zero or more URIs.
* The optional AppliedPolicy element MUST contain zero or more URIs.
* The TransformedDocument element MUST hold a sub\_component .
* The Schemas element MUST hold a sub-component SchemasType.
* The DocumentWithSignature element MUST hold a sub\_component .
* The optional Other element MUST contain zero or more sub-components . Each one MUST satisfy the requirements specified in section PropertyType.
* The VerifyManifestResults element MUST hold a sub-component VerifyManifestResultsType.
* The SigningTimeInfo element MUST hold a sub-component SigningTimeInfoType.
* The VerificationTimeInfo element MUST hold a sub-component VerificationTimeInfoType.
* The ProcessingDetails element MUST hold a sub\_component .
* The SignerIdentity element MUST hold a sub\_component .
* The UpdatedSignature element MUST hold a sub-component UpdatedSignatureType.
* The TimestampedSignature element MUST hold a sub-component UpdatedSignatureType.

### XML Syntax

The XML element SHALL implement in XML syntax the OptionalOutputsVerifyType component.

The OptionalOutputsVerifyType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="OptionalOutputsVerifyType">

<xs:complexContent>

<xs:extension base="dss:OptionalOutputsBaseType">

<xs:sequence>

<xs:element ref="dss:VerifyManifestResults" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:SigningTimeInfo" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:VerificationTimeInfo" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:ProcessingDetails" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:SignerIdentity" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:UpdatedSignature" minOccurs="0" maxOccurs="1"/>

<xs:element ref="dss:TimestampedSignature" minOccurs="0" maxOccurs="1"/>

<xs:element ref="vr:VerificationReport" minOccurs="0" maxOccurs="1"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

Each child element of OptionalOutputsVerifyType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The OptionalOutputsVerifyType XML element SHALL NOT be empty.

### JSON Syntax

The OptionalOutputsVerifyType JSON object SHALL implement in JSON syntax the OptionalOutputsVerifyType component.

The OptionalOutputsVerifyType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-OptionalOutputsVerifyType": {

"type": "object"

"properties": {

"profile": {

"type": "array"

"items": {

"type": "string"

}

}

"policy": {

"type": "array"

"items": {

"type": "string"

}

}

"transformed": {

"type": "object"

"$ref": "#/definitions/dss-TransformedDocument"

}

"schemas": {

"type": "object"

"$ref": "#/definitions/dss-SchemasType"

}

"docWithSignature": {

"type": "object"

"$ref": "#/definitions/dss-DocumentWithSignature"

}

"other": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-PropertyType"

}

}

"result": {

"type": "object"

"$ref": "#/definitions/dss-VerifyManifestResultsType"

}

"signingTimeInfo": {

"type": "object"

"$ref": "#/definitions/dss-SigningTimeInfoType"

}

"verificationTimeInfo": {

"type": "object"

"$ref": "#/definitions/dss-VerificationTimeInfoType"

}

"procDetails": {

"type": "object"

"$ref": "#/definitions/dss-ProcessingDetails"

}

"signerIdentity": {

"type": "object"

"$ref": "#/definitions/saml-NameIdentifierType"

}

"updSignature": {

"type": "object"

"$ref": "#/definitions/dss-UpdatedSignatureType"

}

"timestampedSignature": {

"type": "object"

"$ref": "#/definitions/dss-UpdatedSignatureType"

}

"verificationReport": {

"type": "object"

"$ref": "#/definitions/vr-VerificationReportType"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| AppliedProfile | profile |
| AppliedPolicy | policy |
| TransformedDocument | transformed |
| Schemas | schemas |
| DocumentWithSignature | docWithSignature |
| Other | other |
| VerifyManifestResults | result |
| SigningTimeInfo | signingTimeInfo |
| VerificationTimeInfo | verificationTimeInfo |
| ProcessingDetails | procDetails |
| SignerIdentity | signerIdentity |
| UpdatedSignature | updSignature |
| TimestampedSignature | timestampedSignature |
| VerificationReport |  |

## Element SchemasType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The Schema element MUST hold a sub-component DocumentType.

### XML Syntax

The XML element SHALL implement in XML syntax the SchemasType component.

The SchemasType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="SchemasType">

<xs:sequence>

<xs:element ref="dss:Schema" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

Each child element of SchemasType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The SchemasType XML element SHALL NOT be empty.

### JSON Syntax

The SchemasType JSON object SHALL implement in JSON syntax the SchemasType component.

The SchemasType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-SchemasType": {

"type": "object"

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| Schema |  |

## Element RequestBaseType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The InputDocuments element MUST hold a sub\_component .
* The optional RequestID element MUST hold a string.

### XML Syntax

The XML element SHALL implement in XML syntax the RequestBaseType component.

The RequestBaseType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="RequestBaseType">

<xs:sequence>

<xs:element ref="dss:InputDocuments" minOccurs="0"/>

</xs:sequence>

<xs:attribute name="RequestID" type="xs:string" use="optional"/>

</xs:complexType>

Each child element of RequestBaseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The RequestBaseType XML element SHALL NOT be empty.

### JSON Syntax

The RequestBaseType component has JSON schema represenation as it is not used directly.

## Element ResponseBaseType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The Result element MUST hold a sub\_component .
* The optional RequestID element MUST hold a string.

### XML Syntax

The XML element SHALL implement in XML syntax the ResponseBaseType component.

The ResponseBaseType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="ResponseBaseType">

<xs:sequence>

<xs:element ref="dss:Result"/>

</xs:sequence>

<xs:attribute name="RequestID" type="xs:string" use="optional"/>

</xs:complexType>

Each child element of ResponseBaseType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The ResponseBaseType XML element SHALL NOT be empty.

### JSON Syntax

The ResponseBaseType component has JSON schema represenation as it is not used directly.

## Element TimeSignatureInstructionType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional Type element MUST hold an URI.
* The optional TimeStampTheGivenSignature element MUST hold a boolean.

### XML Syntax

The XML element SHALL implement in XML syntax the TimeSignatureInstructionType component.

The TimeSignatureInstructionType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="TimeSignatureInstructionType">

<xs:complexContent>

<xs:extension base="dss:UpdateSignatureInstructionType">

<xs:attribute name="TimeStampTheGivenSignature" type="xs:boolean" use="optional" default="false"/>

</xs:extension>

</xs:complexContent>

</xs:complexType>

Each child element of TimeSignatureInstructionType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The TimeSignatureInstructionType XML element SHALL NOT be empty.

### JSON Syntax

The TimeSignatureInstructionType component has JSON schema represenation as it is not used directly.

## Element UpdateSignatureInstructionType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional Type element MUST hold an URI.

### XML Syntax

The XML element SHALL implement in XML syntax the UpdateSignatureInstructionType component.

The UpdateSignatureInstructionType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="UpdateSignatureInstructionType">

<xs:attribute name="Type" type="xs:anyURI" use="optional"/>

</xs:complexType>

Each child element of UpdateSignatureInstructionType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The UpdateSignatureInstructionType XML element SHALL NOT be empty.

### JSON Syntax

The UpdateSignatureInstructionType JSON object SHALL implement in JSON syntax the UpdateSignatureInstructionType component.

The UpdateSignatureInstructionType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-UpdateSignatureInstructionType": {

"type": "object"

"properties": {

"type": {

"type": "string"

"format": "uri"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| Type | type |

## Element KeyInfoType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The value element MUST hold base64 encoded binary data.
* The X509Digest element MUST hold a sub\_component .
* The X509SubjectName element MUST hold a string.
* The X509SKI element MUST hold base64 encoded binary data.
* The X509Certificate element MUST hold base64 encoded binary data.
* The KeyName element MUST hold a string.
* The Algorithm element MUST hold an URI.

### XML Syntax

The XML element SHALL implement in XML syntax the KeyInfoType component.

The KeyInfoType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="KeyInfoType">

<xs:choice>

<xs:element name="X509Digest">

<xs:complexType>

<xs:simpleContent>

<xs:extension base="xs:base64Binary">

<xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

</xs:element>

<xs:element name="X509SubjectName" type="xs:string"/>

<xs:element name="X509SKI" type="xs:base64Binary"/>

<xs:element name="X509Certificate" type="xs:base64Binary"/>

<xs:element name="KeyName" type="xs:string"/>

</xs:choice>

</xs:complexType>

Each child element of KeyInfoType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The KeyInfoType XML element SHALL NOT be empty.

### JSON Syntax

The KeyInfoType JSON object SHALL implement in JSON syntax the KeyInfoType component.

The KeyInfoType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-KeyInfoType": {

"type": "object"

"properties": {

"x509Digest": {

"type": "object"

"$ref": "#/definitions/dss-KeyInfoType:X509Digest"

}

"subject": {

"type": "string"

}

"ski": {

"type": "string"

}

"cert": {

"type": "string"

}

"name": {

"type": "string"

}

}

"minProperties": 1

"maxProperties": 1

}

"dss-KeyInfoType:X509Digest": {

"type": "object"

"properties": {

"value": {

"type": "string"

}

"algo": {

"type": "string"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| value | value |
| X509Digest | x509Digest |
| Algorithm | algo |
| X509SubjectName | subject |
| X509SKI | ski |
| X509Certificate | cert |
| KeyName | name |

## Element PropertiesType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The Property element MUST hold a sub-component PropertyType.

### XML Syntax

The XML element SHALL implement in XML syntax the PropertiesType component.

The PropertiesType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="PropertiesType">

<xs:sequence>

<xs:element ref="dss:Property" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

Each child element of PropertiesType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The PropertiesType XML element SHALL NOT be empty.

### JSON Syntax

The PropertiesType JSON object SHALL implement in JSON syntax the PropertiesType component.

The PropertiesType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-PropertiesType": {

"type": "object"

"properties": {

"prop": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-PropertyType"

}

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| Property | prop |

## Element PropertyType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The Identifier element MUST hold an URI.
* The optional Value element MUST hold a sub-component AnyType.

### XML Syntax

The XML element SHALL implement in XML syntax the PropertyType component.

The PropertyType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="PropertyType">

<xs:sequence>

<xs:element name="Identifier" type="xs:anyURI"/>

<xs:element name="Value" type="dss:AnyType" minOccurs="0"/>

</xs:sequence>

</xs:complexType>

Each child element of PropertyType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The PropertyType XML element SHALL NOT be empty.

### JSON Syntax

The PropertyType JSON object SHALL implement in JSON syntax the PropertyType component.

The PropertyType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-PropertyType": {

"type": "object"

"properties": {

"id": {

"type": "string"

}

"value": {

"type": "object"

"$ref": "#/definitions/dss-AnyType"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| Identifier | id |
| Value | value |

## Element VerifyManifestResultsType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The ManifestResult element MUST hold a sub\_component .

### XML Syntax

The XML element SHALL implement in XML syntax the VerifyManifestResultsType component.

The VerifyManifestResultsType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="VerifyManifestResultsType">

<xs:sequence>

<xs:element ref="dss:ManifestResult" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

Each child element of VerifyManifestResultsType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The VerifyManifestResultsType XML element SHALL NOT be empty.

### JSON Syntax

The VerifyManifestResultsType JSON object SHALL implement in JSON syntax the VerifyManifestResultsType component.

The VerifyManifestResultsType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-VerifyManifestResultsType": {

"type": "object"

"properties": {

"result": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-ManifestResult"

}

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| ManifestResult | result |

## Element UseVerificationTimeType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The CurrentTime element MUST hold a boolean.
* The SpecificTime element MUST hold a date/time value.
* The optional Base64Content element MUST contain base64 encoded binary data if present.

### XML Syntax

The XML element SHALL implement in XML syntax the UseVerificationTimeType component.

The UseVerificationTimeType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="UseVerificationTimeType">

<xs:choice>

<xs:element type="xs:boolean" name="CurrentTime" default="false"/>

<xs:element name="SpecificTime" type="xs:dateTime"/>

<xs:element name="Base64Content" type="xs:base64Binary" minOccurs="0" maxOccurs="1"/>

</xs:choice>

</xs:complexType>

Each child element of UseVerificationTimeType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The UseVerificationTimeType XML element SHALL NOT be empty.

### JSON Syntax

The UseVerificationTimeType JSON object SHALL implement in JSON syntax the UseVerificationTimeType component.

The UseVerificationTimeType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-UseVerificationTimeType": {

"type": "object"

"properties": {

"currTime": {

"type": "boolean"

}

"specTime": {

"type": "integer"

"format": "utc-millisec"

}

"b64Content": {

"type": "string"

}

}

"minProperties": 1

"maxProperties": 1

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| CurrentTime | currTime |
| SpecificTime | specTime |
| Base64Content | b64Content |

## Element AdditionalTimeInfoType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The value element MUST hold a date/time value.
* The Type element MUST hold an URI.
* The optional Ref element MUST hold a string.

### XML Syntax

The XML element SHALL implement in XML syntax the AdditionalTimeInfoType component.

The AdditionalTimeInfoType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="AdditionalTimeInfoType">

<xs:simpleContent>

<xs:extension base="xs:dateTime">

<xs:attribute name="Type" type="xs:anyURI" use="required"/>

<xs:attribute name="Ref" type="xs:string" use="optional"/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

Each child element of AdditionalTimeInfoType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The AdditionalTimeInfoType XML element SHALL NOT be empty.

### JSON Syntax

The AdditionalTimeInfoType JSON object SHALL implement in JSON syntax the AdditionalTimeInfoType component.

The AdditionalTimeInfoType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-AdditionalTimeInfoType": {

"type": "object"

"properties": {

"value": {

"type": "integer"

"format": "utc-millisec"

}

"type": {

"type": "string"

"format": "uri"

}

"ref": {

"type": "string"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| value | value |
| Type | type |
| Ref | ref |

## Element VerificationTimeInfoType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The VerificationTime element MUST hold a date/time value.
* The AdditionalTimeInfo element MUST hold a sub-component AdditionalTimeInfoType.

### XML Syntax

The XML element SHALL implement in XML syntax the VerificationTimeInfoType component.

The VerificationTimeInfoType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="VerificationTimeInfoType">

<xs:sequence>

<xs:element name="VerificationTime" type="xs:dateTime"/>

<xs:element ref="dss:AdditionalTimeInfo" minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

Each child element of VerificationTimeInfoType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The VerificationTimeInfoType XML element SHALL NOT be empty.

### JSON Syntax

The VerificationTimeInfoType JSON object SHALL implement in JSON syntax the VerificationTimeInfoType component.

The VerificationTimeInfoType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-VerificationTimeInfoType": {

"type": "object"

"properties": {

"verificationTime": {

"type": "integer"

"format": "utc-millisec"

}

"additionalTimeInfo": {

"type": "array"

"items": {

"type": "object"

"$ref": "#/definitions/dss-AdditionalTimeInfoType"

}

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| VerificationTime | verificationTime |
| AdditionalTimeInfo | additionalTimeInfo |

## Element DetailType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The optional Code element MUST hold an URI.
* The optional Message element MUST hold a sub-component InternationalStringType.
* The optional Base64Content element MUST contain base64 encoded binary data if present.
* The Type element MUST hold an URI.

### XML Syntax

The XML element SHALL implement in XML syntax the DetailType component.

The DetailType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="DetailType">

<xs:sequence>

<xs:element name="Code" type="xs:anyURI" minOccurs="0"/>

<xs:element name="Message" type="dss:InternationalStringType" minOccurs="0"/>

<xs:element name="Base64Content" type="xs:base64Binary" minOccurs="0" maxOccurs="1"/>

</xs:sequence>

<xs:attribute name="Type" type="xs:anyURI" use="required"/>

</xs:complexType>

Each child element of DetailType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The DetailType XML element SHALL NOT be empty.

### JSON Syntax

The DetailType JSON object SHALL implement in JSON syntax the DetailType component.

The DetailType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-DetailType": {

"type": "object"

"properties": {

"code": {

"type": "string"

}

"msg": {

"type": "object"

"$ref": "#/definitions/dss-InternationalStringType"

}

"b64Content": {

"type": "string"

}

"type": {

"type": "string"

"format": "uri"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| Code | code |
| Message | msg |
| Base64Content | b64Content |
| Type | type |

## Element SigningTimeInfoType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The SigningTime element MUST hold a date/time value.
* The optional SigningTimeBoundaries element MUST hold a sub\_component .
* The optional LowerBoundary element MUST hold a date/time value.
* The optional UpperBoundary element MUST hold a date/time value.

### XML Syntax

The XML element SHALL implement in XML syntax the SigningTimeInfoType component.

The SigningTimeInfoType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="SigningTimeInfoType">

<xs:sequence>

<xs:element name="SigningTime" type="xs:dateTime"/>

<xs:element name="SigningTimeBoundaries" minOccurs="0">

<xs:complexType>

<xs:sequence>

<xs:element name="LowerBoundary" type="xs:dateTime" minOccurs="0"/>

<xs:element name="UpperBoundary" type="xs:dateTime" minOccurs="0"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

Each child element of SigningTimeInfoType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The SigningTimeInfoType XML element SHALL NOT be empty.

### JSON Syntax

The SigningTimeInfoType JSON object SHALL implement in JSON syntax the SigningTimeInfoType component.

The SigningTimeInfoType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-SigningTimeInfoType": {

"type": "object"

"properties": {

"signingTime": {

"type": "integer"

"format": "utc-millisec"

}

"signingTimeBounds": {

"type": "object"

"$ref": "#/definitions/dss-SigningTimeInfoType:SigningTimeBoundaries"

}

}

}

"dss-SigningTimeInfoType:SigningTimeBoundaries": {

"type": "object"

"properties": {

"lowerBound": {

"type": "integer"

"format": "utc-millisec"

}

"upperBound": {

"type": "integer"

"format": "utc-millisec"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| SigningTime | signingTime |
| SigningTimeBoundaries | signingTimeBounds |
| LowerBoundary | lowerBound |
| UpperBoundary | upperBound |

## Element UpdatedSignatureType

**Semantics**

Below follows a list of the sub-components that MAY be present within this component:

* The SignatureObject element MUST hold a sub\_component .
* The optional Type element MUST hold an URI.

### XML Syntax

The XML element SHALL implement in XML syntax the UpdatedSignatureType component.

The UpdatedSignatureType XML element SHALL be defined as in XML Schema file [FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE XSD], and is copied below for information.

<xs:complexType name="UpdatedSignatureType">

<xs:sequence>

<xs:element ref="dss:SignatureObject"/>

</xs:sequence>

<xs:attribute name="Type" type="xs:anyURI" use="optional"/>

</xs:complexType>

Each child element of UpdatedSignatureType XML element SHALL implement in XML syntax the sub-component that has a name equal to its local name.

The UpdatedSignatureType XML element SHALL NOT be empty.

### JSON Syntax

The UpdatedSignatureType JSON object SHALL implement in JSON syntax the UpdatedSignatureType component.

The UpdatedSignatureType JSON object SHALL be defined as in JSON Schema file [JSON SCHEMA FILE NAME] whose location is detailed in clause [CLAUSE FOR LINK TO THE JSON SCHEMA FILE], and is copied below for information.

"dss-UpdatedSignatureType": {

"type": "object"

"properties": {

"sigObj": {

"type": "object"

"$ref": "#/definitions/dss-SignatureObject"

}

"type": {

"type": "string"

"format": "uri"

}

}

}

Each property in the JSON schema above SHALL implement in JSON syntax one sub-component of InputDocuments component as shown in the table below.

|  |  |
| --- | --- |
| Component | Implementing JSON member name |
| SignatureObject | sigObj |
| Type | type |

# Processing Model

Here we place the many processing step model variations from 1.0 as they fit …

# JSON Format

Here we place the JSON extended world view on DSS AND\_REMOVE\_<==\_WHEN\_FINISHED.

JSON, as described in [[RFC7159](#refRFC7159)], defines a text format for serializing structured data. Objects are serialized as an unordered collection of name/value pairs.

JSON does not define any semantics around the name/value pairs that make up an object, nor does it define an extensibility mechanism for adding control information to a payload.

DSS’s JSON format extends JSON by defining general conventions for name/value pairs that annotate a JSON object, property or array. DSS defines a set of canonical annotations for control information such as ids, types, and links, and custom annotations MAY be used to add domain-specific information to the payload.

Annotations are used in JSON to capture control information that cannot be predicted as well as a mechanism to provide values where a computed value would be wrong.

## JSON – Type Base64DataType

The generic entity Base64DataType is defined in 3.1 [Type Base64DataType](#_Type_Base64DataType).

WE\_INSERT\_SYSTEMATIC\_NUMERATED\_SPEC\_TABLE\_CAPTIONS Element name mapping table:

|  |  |
| --- | --- |
| Element | JSON Member Name |
| ConceptName | A\_SIMILAR\_THING |

|  |  |
| --- | --- |
| Element | JSON member name |
| ValueOf(InstanceOf(Base64DataType)) | value |
| AttRefUri | attRef |
| Id | ID |
| IdRef | IDREF |
| MimeType | mimeType |

JSON sample:

"b64Data" : {  
 "value" : "VGVzdERvY3VtZW50",  
 "mimeType" : "application/text",  
 "ID" : "contentId-8847908085513926610"  
}

The elements ID and IDREF have no special role in the JSON syntax.

« The uniqueness of ID and the referential integrity of the ID / IDREF pair MUST be ensured by the implementation. » [DJS-9.1-1]

# XML Format

Here we place the XML world view on DSS AND\_REMOVE\_THIS\_SENTENCE\_WHEN\_FINISHED.

## XML – Type Base64DataType

The generic entity Base64DataType is defined in 3.1 [Type Base64DataType](#_Type_Base64DataType).

WE\_INSERT\_SYSTEMATIC\_NUMERATED\_SPEC\_TABLE\_CAPTIONS Element name mapping table:

|  |  |
| --- | --- |
| Element | XML Entity |
| ConceptName | A\_SIMILAR\_THING |

|  |  |
| --- | --- |
| Element | XML entity |
| ValueOf(InstanceOf(Base64DataType)) | TEXT(InstanceOf(Base64DataType)) |
| AttRefUri | AttRefURI |
| Id | ID |
| IdRef | IDREF |
| MimeType | MimeType |

XML schema snippet defining Base64DataType:

<xs:complexType name="Base64DataType">  
 <xs:simpleContent>  
 <xs:extension base="xs:base64Binary">  
 <xs:attribute name="MimeType" type="xs:string" use="optional"/>  
 <xs:attribute name="AttRefURI" type="xs:anyURI" use="optional"/>  
 <xs:attribute name="ID" type="xs:ID" use="optional"/>  
 <xs:attribute name="IDREF" type="xs:IDREF" use="optional"/>  
 </xs:extension>  
 </xs:simpleContent>  
</xs:complexType>

The elements ID and IDREF take advantage of XML’s ID mechanism.

# DSS-Defined Identifiers

The following sections define various URI-based identifiers. Where possible an existing URN is used to specify a protocol. In the case of IETF protocols the URN of the most current RFC that specifies the protocol is used (see **[RFC 2648]**). URI references created specifically for DSS have the following stem:

urn:oasis:names:tc:dss:1.0:

## Signature Type Identifiers

The following identifiers MAY be used as the content of the <SignatureType> optional input (see section 3.5.1).

### XML Signature

* **URI:** urn:ietf:rfc:3275
* This refers to an XML signature per **[XMLDSIG]**.

### XML TimeStampToken

* **URI:** urn:oasis:names:tc:dss:1.0:core:schema:XMLTimeStampToken
* This refers to an XML timestamp containing an XML signature, per section 5.1.

### RFC 3161 TimeStampToken

* **URI:** urn:ietf:rfc:3161
* This refers to an XML timestamp containing an ASN.1 TimeStampToken, per **[RFC 3161]**.

### CMS Signature

* **URI:** urn:ietf:rfc:3369
* This refers to a CMS signature per **[RFC 3852]** or prior versions of CMS.

### PGP Signature

* **URI:** urn:ietf:rfc:2440
* This refers to a PGP signature per **[RFC 2440]**.

# Conformance

## Conformance as a DSS version 2.0 document

To ease communication and subsequent resolution of any specific partial conformance violation, the preceding chapters already provide minimal requirements, that a specific instance component must fulfill, to permit conformance of the complete DSS version 2.0 document.

### Conformance for XML format

The following clause offers a simple three step process, to either prove or disprove the conformance of a complete XML document (formulated in terms specific to that implementation language) to this version of DSS:

∇ An XML document instance conforms to this specification as a DSS document if it meets all of the following three conditions:

1. Is well-formed XML.
2. Consists of a single dss:whatever element instance as defined in   
   the namespace http://docs.oasis-open.org/dss-x/ns/dss-core/v2.0/dss.
3. Is valid XML.

∆ [DSS-5.1.1-1]

### Conformance for JSON format

The following clause offers a simple COUNT\_ME step process, to either prove or disprove the conformance of a complete JSON document (formulated in terms specific to that implementation language) to this version of DSS:

∇ A JSON document instance conforms to this specification as a DSS document if it meets all of the following COUNT\_ME conditions:

1. Is valid JSON
2. Other COUNT\_ME minus 1 criteria …

∆ [DSS-5.1.2-1]

1. Acknowledgments

The following individuals were members of the OASIS DSS-X Technical Committee during the creation of this specification and their contributions are gratefully acknowledged:

Andreas Kuehne, Individual

Andreas Reiter, A-SIT, Zentrum fuer sichere Informationstechnologie Austria

Chet Ensign, OASIS

Detlef Huehnlein, Individual

Ernst Jan van Nigtevecht, Sonnenglanz Consulting

Ezer Farhi, DocuSign, Inc.

Herbert Leitold, A-SIT, Zentrum fuer sichere Informationstechnologie Austria

Juan Cruellas, Departamento de Arquitectura de Computadores, Univ Politecnica de Cataluna

Mark Klamerus, Individual

Michael Yatsko, DocuSign, Inc.

Pim van der Eijk, Sonnenglanz Consulting

Robin Cover, OASIS

Stefan Hagen, Individual

1. 1. Use of Exclusive Canonicalization

Exclusive Canonicalization of dereferenced and transformed data can be achieved by appending exclusive canonicalization as the last transform in the <ds:Transforms> element of TransformedData or DocumentHash.

In the case of Document being used this can be done by adding exclusive canonicalization as the last transform in the ds:Transforms of a SignedReference pointing to that Document.

By doing this the resulting data produced by the chain of transforms will always be octet stream data which will be hashed without further processing on a <ds:Reference> level by the server as indicated by basic processing section 4.3.1 step 1 b. and c.

Another possibility to apply exclusive canonicalization on <ds:Reference> level is the freedom given to servers to apply additional transforms to increase robustness. This however implies that only trustworthy transformations are appended by a server.

As in section 4.3.1 step 1 b an implementation can choose to use exclusive canonicalization: "... Transforms are applied as a server implementation MAY choose to increase robustness of the Signatures created. These Transforms may reflect idiosyncrasies of different parsers or solve encoding issues or the like. ..."

In such a case that the exclusive canonicalization is to be included in the ds:Transforms as well (cf. section 4.3.1 step 1.d.v.)

The standards default is however in line with [XMLDSIG] as indicated in the Note in section 4.3.1 step 1 b.

However after the server formed a <ds:SignedInfo> (section 4.3.1 step 3.) this information to be signed also needs to be canonicalized and digested, here [XMLDSIG] offers the necessary element <ds:CanonicalizationMethod> directly and can be used to specify exclusive canonicalization.

* 1. More Complex Response Example

To further explain the use of the Response element which is useful in cases where the DSS server is not able to respond with a special response type a more complex example is given in the following paragraph.

Consider for example a client sends a SignRequest to a service that only supports VerifyRequests over plain HTTP (as opposed to protocols where some information could be derived from the header). As the service does not support SignRequest's it has to either generate a VerifyResponse with a "bad message" result or fail at the HTTP layer. In the former case, the client will receive a response that does not correspond semantically to the request - it got a VerifyResponse to a SignRequest. This leaves both parties thinking that the other one is at fault.

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Editor** | **Changes Made** |
| [Rev number] | [Rev Date] | Andreas Kuehne and Stefan Hagen | Initial Draft version with feedback from the TC |