# Evolving the DSS-X standard

Andreas Kühne[1]

**Abstract:**

This document describes the adoption of an existing specification (for signature creation and validation) to new challenges both in signature-specific and general technical requirements. The major work item is the need to support multiple interface description syntaxes. This document also discusses an approach of automatic document generation to provide multiple artefacts in a consistent and timely manner.

This contribution wants to outline a way to maintain specifications in a changing landscape of requirements.

**Keywords:** signature creation, signature verification, JSON, XML

## 1 Introduction

The [Di18a] specification on signature creation and validation became official OASIS standard in April 2007. It defines the corresponding methods using XML schema referencing other well-known schemes (e.g. [XM18]). To provide flexibility for extensions the editors used several XML schema-specific features (e.g. the 'mixed' attribute).

To reflect further development both in general and in the area of signature creation and verification, the OASIS DSS-X technical committee started the effort to produce a version 2.0 of the standard 2016. Section 2 outlines the signature related changes. A major driver for the new version is the ubiquitous use of JSON. But the existing XML-based systems should not be cut off from further developments. Therefore, a significant effort was invested to support multiple transport syntaxes in parallel while using the same syntactical model. This approach is discussed in Section 3.

To ensure the general adoption of a standard it is recommended to provide additional supportive material that eases the practical use of it. This can be a sample implementation, a conformance testbed or an interactive user interface to try the specification at well-known platforms (e.g. SwaggerHub[2]) Section 4 closes this contribution by summarising the main aspects and providing an outlook on possible future developments.

[1] trustable Ltd (Germany), Standardization, Gartenheimstr. 39C, Hannover, 30659, kuehne@trustable.de
[2] https://app.swaggerhub.com/apis/OASIS.Open/oasis-dss_2_0/0.1

## 2 Changes in core functionality

The main changes of this version of the DSS-X core document [Di18b] compared to version 1.0 are:

- Process the set of comments and bug reports arrived since version DSS 1.0 became standard.
- Inclusion of requirements that became known only after publication of version 1.0.
- Simplification of the core schema, e.g. by dropping elements seldom used.
- Integration of the 'Asynchronous Processing Profile' [As18a] into the core
- Support [As18b].
- The definition of a XML timestamp format in [Di18a], section 5.1 will not be upgraded to [Di18b].

To support implementers and to ease the use of the protocol with common frameworks, the following list of requirements were respected:

- One unique object model for all transport syntaxes.
- Define type and cardinalities of `OptionalInputs*` and `OptionalOutputs*` child elements explicitly.
- Rearrange sequences and choices to produce a strongly typed object model
- Extract basic types into a separate XML schema to support their use in non-signature related specifications.

The provided schemes of DSS-X version 2 reflect these requirements. The XML schemes of version 1 and 2 share many similarities but are not compatible. These group of changes can be considered as 'usual business' for a committee maintaining a specification and don't require an adoption of the specification creation process.

## 3 Multi Syntax approach

### 3.1 Challenges

The formerly dominant [SO18] solution stack lost its leading role for newly designed interfaces. Nevertheless, there will be a significant implementation base in productive environments for years to come. The success of [Th18]-based interfaces in the last years is quite impressive. It took over the role as preferred solution and is supported by many design and implementation tools. But, as seen with SOAP, new trends may introduce new approaches in the future. Specific technical requirements (e.g. low bandwidth mobile connections) to support special purpose solutions (e.g. the compact [AB18] format) could also be a driver for change.

### 3.2 Solution path

To provide a solution path for this set of potential challenges, the TC did choose a comprehensive approach: Do not to limit syntax support to a set of currently relevant ones (XML & JSON) but to separate the semantic of an interface from the implementation syntax. The DSS-X 2.0 specification defines a sematic model for each component that is mapped to XML and JSON, but offers the mapping to additional syntaxes.

Different syntaxes support distinct sets of features. Therefore, only a common denominator of features can be used. The DSS 1.0 version supports a set of data transport variants, most of them are XML-syntax specific. Base64 encoded data offers the most versatile way to transport documents and signatures. This transport mode can be found in most transport syntaxes and was therefore selected as the preferred solution. The data volume overhead is a drawback but the advantages of Base64 encoded data are worth the performance penalty.

Several problems and drawbacks arise when leaving the well-known sphere of XML semantic and syntax. The aspects listed in the following table needed special consideration:

- Replace `xs:any` with an enumeration of possible types. If that is not feasible, use base64 blobs as a fallback.
- Avoid the use of XML specifics (like e.g. mixed content).
- Provide namespace / URI for XPath evaluation explicitly.

The aspects and the applied solutions are discussed in the following chapters.

### 3.3 Circumventing xs:any

The XML schema type ‘any’ allows an object to contain arbitrary structures. This comes handy for writers of specifications as an extension point because the structures transported do not need to be defined upfront. But this advantage at the specification stage comes with a price at the implementation stage. The structures intended to be supported by a client or a server system MUST be known to be implementable. But the usual tools for schema support leave the task of handling the content of an any type to the developer. Without extensive testing problems with unexpected content may occur at runtime, even while using typed languages.

The OptionalInputs element (of DSS version 1.0) makes use of `xs:any`. The replacement component OptionalInputsVerify (of DSS-X version 2.0) defines its child elements and their cardinality explicitly. When using additional profiles, the relevant components of the core schema can be redefined using the XML schema’s ‘redefine’ element or JSON schema’s ‘allOf’.

Another usage scenario for `xs:any` is the transport of unknown data objects. A sample use case is the Property component. This component is intended to contain signature attributes of unknown structure. In DSS-X version 2.0 the `xs:any` type is replaced by a structure containing base64-encoded data and meta data. When using XML as the transport syntax this seems to be a disadvantage. But direct XML fragment copying may introduce namespace problems and security concerns. Most importantly, the cherry-picking of transport syntax features would inhibit a transport independent object model, both on the client and the server side. More complex programming and testing would be inevitable.

### 3.4 Substituting the 'mixed' schema attribute

Mixing sub-elements and text within a single element is a great advantage of XML. But when XML is applied for serializing an object model this 'markup language' feature is of little use. Other serialization syntaxes (like JSON) don't support such a feature. There is the need to substitute the '`mixed`' construct to become syntax independent. The substitution is done by removing the mixed attribute and introduce an additional '`value`' element to contain the textual content.

### 3.5 Introducing the `NsPrefixMappingType` component

Namespaces are an outstanding feature of the XML world. A replacement is required for all syntaxes that don't such a feature. The use of naming conventions and prefixes are common to avoid naming collisions. A special challenge is the use of XPath expressions as elements. The XPath expression itself is represented as a simple string. But the expression may depend on namespace/prefix mappings that are defined within the namespace context of the XML element. The `NsPrefixMappingType` component (of DSS-X version 2.0) represents the required namespace/prefix mapping. It is recommended to use this element for XML syntax, too. This simplifies the handling on the consumer side and circumvents problems with namespace prefix assignments handled by web frameworks.

### 3.6 Imported XML schemes

A special challenge is imposed by the imported schemes, like the **[XM18]** scheme, that uses features not supportable by the mentioned 'multi-syntax' approach. The most obvious restrictions are:

- The complexType may contain mixed content (child elements **and** text). This concept is not supported by JSON. The workaround for this limitation is to drop the 'mixed' attribute and to introduce a 'value' element.
- The '`choice`' construct is mapped in an untyped way by Java's JAXB framework. Therefore, the '`choice`' element is changed to a '`sequence`'.

- The 'any' type is replaced by a base64 encoded blob.
- The option to provide arbitrary namespace / prefix mappings to support the evaluation of XPath expression is not available in e.g. JSON syntax. Therefore an element mapping prefixes to namespaces (of type '`dsb:NsPrefixMappingType`') is added.

To apply the necessary changes to the imported schemes the XML schema language provides the '`override`' functionality to change existing schemes. But Java's JAXB framework's schema compiler does not support '`override`' so the adapted schemes are provided alongside DSS-X core schemes.

### 3.7 Automation requirements

The interface descriptions for different syntaxes are expected to be available in their specific formats (XML Schema for XML, JSON Schema for JSON, modules for [Ab18]) and need to be kept aligned with the specification document. To provide a reliable quality of the documents and to minimize the human effort, the DSS-X TC uses a single-source approach for parts of the specification and the schemes. The semantic requirements are formulated using a restricted set of XML Schema. Based on this information a generator produces the depending schema documents and replaces the related sections in the specification.

To support specific syntax features or common usage patterns the XML representation of the semantics is extended. Using this extension mechanism e.g. the usually short tag names of JSON are provided.

The generating of the dependent artefacts (e.g. schema files) is straight forward and can be performed without user interaction. The tooling set also allows the direct editing of 'editorial' parts within the generated parts of the specifications and preserves this content over repeated generation processes. This gives the editor the opportunity of textual enrichment of generated sections (e.g. general component comment, (non-)normative sections, explanations of element, syntax specific comments).

The specification document consists of both manually edited and generated sections. To support a smooth editing process preserving the user input even in case of changed semantics the editor's contribution must be preserved, e. g. in a database. The stored content is not just input for the assembly of a specification document, it also proved to be useful for the generation of interface descriptions like the Open API Specification [Op18].

## 4 Summary and Outlook

Ten years after becoming official standard the [Di18a] specification deserves a reengineering to align to the changed requirement landscape. The signature creation and verification-related topics of the core specification were manageable. The far bigger challenge was the support for the changes of the technical landscape. The chosen ‘multi syntax approach’ promises the required flexibility for the next decade. The required automation functionalities will support the editor and ensure a consistent high level of quality of the different output documents.

The automatic generation process will be extended to produce additional artefacts in a reliable manner to minimize human effort while ensuring consistency for all output formats.

The forthcoming re-working of the existing profiles will benefit from the existing tooling.

Regardless of the use of JSON as a transport syntax the handling of JSON signatures will not be covered by the core specification. A dedicated profile will address signatures e.g. conformant to [JS18].

## Bibliography

| | |
|---|---|
| [Ab18] | Abstract Syntax Notation One (ASN.1): Specification of basic notation, https://www.itu.int/rec/T-REC-X.680-200811-I/en, accessed: 04.11.2018 |
| [As18a] | ASN.1 encoding rules: Specification of Packed Encoding Rules (PER), https://www.itu.int/rec/T-REC-X.691-200811-I/en, accessed: 04.11.2018 |
| [As18a] | Asynchronous Processing Abstract Profile of the OASIS Digital Signature Services Version 1.0, http://docs.oasis-open.org/dss/v1.0/oasis-dss-profiles-asynchronous_processing-spec-v1.0-os.html, accessed: 04.11.2018 |
| [Th18] | The JavaScript Object Notation (JSON) Data Interchange Format, https://tools.ietf.org/html/rfc8259, accessed: 04.11.2018 |
| [Op18] | OpenAPI Specification, https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md, accessed: 04.11.2018 |
| [Di18a] | Digital Signature Service Core Protocols, Elements, and Bindings Version 1.0, http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.html, accessed: 04.11.2018 |
| [Di18b] | Digital Signature Service Core Protocols, Elements, and Bindings Version 2.0, http://docs.oasis-open.org/dss-x/dss-core/v2.0/csprd01/dss-core-v2.0-csprd01.pdf, accessed: 04.11.2018 |
| [JS18] | JSON Web Signature (JWS, https://tools.ietf.org/html/rfc7515, accessed: 04.11.2018 |
| [SO18] | SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), https://www.w3.org/TR/soap12/, accessed: 04.11.2018 |
| [As18b] | Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf, accessed: 04.11.2018 |
| [XM18] | XML-Signature Syntax and Processing, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/, accessed: 04.11.2018 |