Open Score Format

Packaging Specification

Version 1.0

Date: 15/9/2009

Revision History

Date	Version	Description	Author
15/12/08	0.9	Public beta	Mark Olleson, Yamaha R&D Centre London
10/9/09	1,0	OSF 1.0	Mark Olleson, Yamaha R&D Centre London

Acknowledgements

The following organizations have contributed to the development of the Open Score Format

HAL•LEONARD®

The Music Sales Group

makemusic®

Recordare

YAMAHA

License

Table of Contents

Open Score Format - Packaging Specification

1. Introduction

1.1 Purpose

This document specifies the format of Open Score Format packages and their associated metadata.

1.2 Scope

The Open Score Format is an open and non-proprietary file format for digital scores and associated media objects and metadata. The Open Score Format is intended to be used for delivering digital scores to end users, as an interchange format between applications that create or use digital scores and for archive purposes.

Digital score content in an Open Score Format package is represented in MusicXML. Restricted subsets of MusicXML permitted for particular types of content types are called *application profiles*. The initial version of the Open Score Format contains a single profile for PVG content. This is defined by the Open Score Format PVG Application Profile [10].

This document defines the layout of the Open Score Format packages and associated metadata, the catalogue metadata for the work contained within the package and the manner in which packages contents can be digitally signed.

1.3 Intended Audience

This document is primarily intended for developers of music application software that creates, manipulates, displays or sells digital score content.

1.4 Definitions, Acronyms and Abbreviations

- **Open Score Format Application**: An application that is capable of reading, writing or otherwise processing Open Score Format packages.
- **Open Score Format Package**: A single file system object that contains all of the necessary data and metadata associated with one or more renditions of a digital score.
- **PVG**: Piano Voice Guitar (for musical score)

1.5 Use of Requirements Levels

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC 2119 [11].

1.6 Conventions

1.6.1 Typographic Conventions

Fragments of XML and file-paths appear in a blue mono-spaced type-face:

- XML fragment: `<dc:title>`
- File-path: `META-INF/Manifest.xml`

Multi-line XML excepts appear with a grey background and syntax-highlighted text:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

1.6.2 *Schema Prefixes and Namespace URIs*

Prefix	Schema	Namespace URI
`osfm`	Manifest	`http://openscoreformat.sourceforge.net/osf/manifest`
`osfmeta`	Metadata	`http://openscoreformat.sourceforge.net/osf/metadata`
`dc`	Simple Dublin Core	`http://purl.org/dc/elements/1.1/`
`dcterms`	Qualified Dublin Core	`http://purl.org/dc/terms/`
`ds`	XML-SIG digital signatures	`http://www.w3.org/2000/09/xmldsig#`

Table 1: XML Namespaces and prefixes used in this document

1.6.3 *Schema Locations*

Prefix	Schema	Schema Document URI
`osfm`	Manifest	`http://openscoreformat.sourceforge.net/1.0/manifest.xsd`
`osfmeta`	Metadata	`http://openscoreformat.sourceforge.net/1.0/metadata.xsd`
<none>[1]	Container	`http://www.musicxml.org/xsd/container.xsd`
`dc`	Simple Dublin Core	`http://dublincore.org/schemas/xmls/qdc/2008/02/11/dc.xsd`
`dcterms`	Qualified Dublin Core	`http://dublincore.org/schemas/xmls/qdc/2008/02/11/dcterms.xsd`
<none>[2]	OSF PVG Application Profile	`http://www.musicxml.org/xsd/osfpvg.xsd`
<none>	MusicXML 2.0 Strict	`http://www.musicxml.org/xsd/musicxml.xsd`

Table 2: Location of W3C XML Schemas used in this document

1.7 References

[1] OEBPS Container Format 1.0

[2] Dublin Core

[3] Resource Description Framework

[4] RFC2048: Multipurpose Internet Mail Extensions (MIME) Part 2: Media Types

[5] Resource Description Framework (RDF)

[6] RDF/XML Syntax Specification (Revised).

[1] The **`container.xsd`** schema has no namespace for backward compatibility with MusicXML containers which have a file with the same name and content model.

[2] The **`osfpvg.xsd`** schema has no namespace due to it being derived from the MusicXML Strict 2.0 Schema – which in turn has historically not used namespaces due to forward- and backwards-compatibility requirements

[7] RFC3987: Internationalized Resource Identifiers (IRIs)

[8] ZIP File Format Specification v6.3.2

[9] XML Signature Syntax and Processing Specification (Second Edition)

[10] Open Score Format PVG Application Profile Specification[11] RFC2119: Keywords for use in RFCs to indicate requirements levels

[12] XML 1.1 Extensible Markup Specification

[13] Dublin Core Metadata Initiative: Dublin Core Qualifiers

[14] RFC4646: Tags for Identification of Languages

[15] RFC2141: Uniform Resource Name Syntax

[16] RFC3187: Using International Standard Book Numbers as Uniform Resource Names

[17] RFC4122: A Universally Unique Identifier (UUID) URN Namespace

[18] W3C Note: Date and Time Formats

[19] DCMI Period Encoding Scheme: specification of the limits of a time interval, and methods for encoding this in a text string

[20] Dublin Core Metadata Initiative: Using Dublin Core – The elements

1.8 Overview

The remainder of the document is structured as follows:

- Section 2 describes the design philosophy of the Open Score Format
- Section 3 defines the format of the package and its contents
- Section 4 describes how to digital sign packages
- Section 5 describes the use of metadata in Open Score Format packages
- Section 6 describes the OSF Package Validation Tool
- Section 7 is the appendix

2. Open Score Format Design Philosophy

2.1 Introduction

The objective of the Open Score Format is to provide an open and non-proprietary file format for digital scores and their associated metadata that allows reliable interchange between applications. The openness of the file format is significant as data often outlives the applications that creates it.

The *Open Score Format base-line profile* defines minimum requirements for interoperability.

2.2 Open Score Format Base-line Profile

The *Open Score Format base-line profile* defines the following:

- A standardized container file format for scores, associated media objects and metadata
- A specification of the minimum amount of catalogue metadata required for describing the package - the *Open Score Format Base-line metadata profile*
- A subset of MusicXML used to represent the digital score

Digital scores in an Open Score Format package conform to one of a number of *Digital Score Application Profiles.*

2.3 Digital Score Application Profiles

MusicXML allows a considerable degree of flexibility to accommodate almost any type of score annotation and practice commonly used in Western European tradition scores. Building and testing tools to accommodate all of these features is a daunting task.

The Open Score Format uses *Digital Score Application Profiles* to define a subset of MusicXML score elements and features that are required for a particular genre of score. The result is a reduction in the effort required to implement and test software that makes use of these scores.

The first release of the Open Score Format specification introduces a single *Digital Score Application Profile* for PVG content – the most prominent commercial content type. It is envisaged that additional profiles could be added at a later date.

2.4 Use of Existing Open Standards

The design of the Open Score Format Packaging Specification has made use of existing open standards and specifications for content packaging and metadata where practical:

- The Open Score Format Packages Specification draws upon elements of the OEBPS Container Format (OCF) specification [1]. OCF is a packaging format for eBooks.

  This package layout is a superset of that employed by MusicXML 2.0 for its container format. This means that software that understands MusicXML 2.0 containers should also be capable of reading Open Score Format packages, although MusicXML 2.0 containers are not valid Open Score Format packages.
- The Dublin Core [2] is used as the basis of the metadata vocabulary in Open Score Format. The Dublin Core is a standardized vocabulary for describing resources' catalogue data. It is widely used for hard-goods such as books as well as for digital resources such as video and audio.
- XML is used throughout for metadata files. XML is inherently extensible.
- The *XML Signature Syntax and Processing Specification* [9] is used to implement package signing.

2.5 Extensibility

The Open Score Format is purposefully designed to provide opportunity for extensibility:

- Open Score Format packages **MAY** contain multiple, application-specific renditions of content. These can be used either to enhance the default rendition or provide an alternate rendition. The use of renditions is described in §3.2
- The Open Score Format *Base-line Metadata Profile* provides an extension mechanism that allows particular applications to add application-specific data to that which is already included in the package

2.6 Expectations of Interoperability

Whilst the Open Score Format is designed to allow extensibility, it is important that packages remain interoperable with all Open Score Format reading applications.

Where an extension is used, the package **MUST** provide base-line functionality to applications that understand the *Open Score Format base-line profile.*

3. Packaging

3.1 Introduction

An Open Score Format package is a container for digital scores, media objects such as audio or video associated with the score and metadata. It **MUST** contain precisely one content title that can be represented using a single MusicXML score file.

Examples of a single content title include:

- A song-sheet for a single song
- A single musical work with several smaller sub-parts such as a concerto with several movements.
- A book of exercises or studies that are sold as a single unit.
- A method book comprising of score excerpts and instructional text.

An Open Score Format package **SHOULD NOT** include:

- Compendiums or collections of separate works – for instance a song-book containing transcriptions of the tracks from a CD album.

Instead, separate Open Score Format packages should be used for each part and metadata should be used to describe the relationship between them.

3.2 Renditions

An Open Score Format package **MAY** include multiple *renditions* of the content title. Renditions allow the content title to be presented in multiple data formats or with enhancements for particular applications.

In order to ensure that the expectations of interoperability between applications are met (see §2.6), a *default rendition* of the score that complies with one of the *digital score application profiles* **MUST** be provided.

Renditions provided in addition to the *default rendition* are called *alternate renditions* and their use is **OPTIONAL**. Potential uses include:

- A pre-rendered edition of the score (for instance in PDF or Postscript) suitable for high quality printing.
- Providing the original score engraving in the native file-format of the tool in which it was originally engraved. This is particularly appropriate for archive applications.
- Supporting applications such as Yamaha's Digital Music Notebook that provide an enhancement of the score with additional media objects such as video, audio and MIDI.

One or renditions **MAY** reference each object within a package.

3.3 Package Format

An Open Score Format package is a ZIP file [4] containing the directory structure shown below.

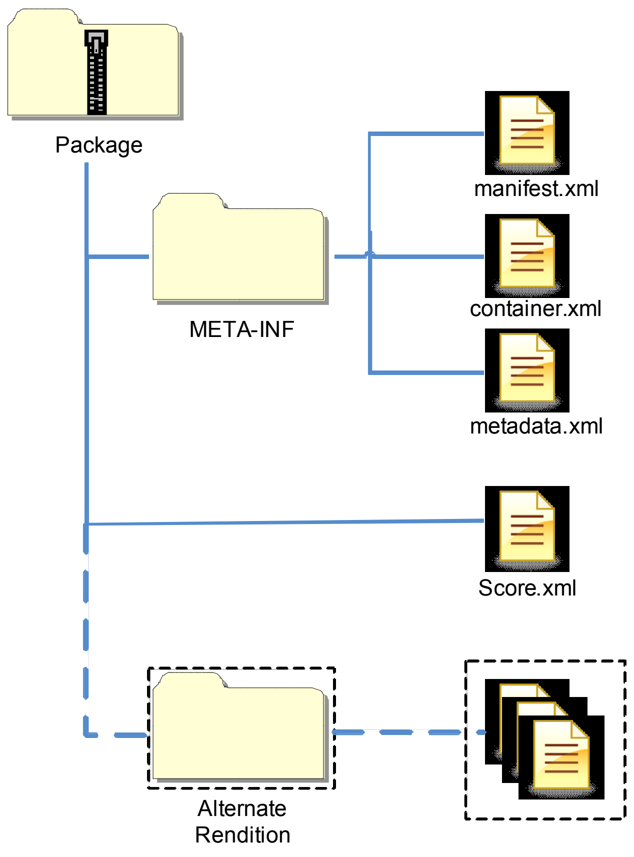

Figure 1: Directory structure of an Open Score Format Package

The diagram above shows high-level view of the items that appear in an Open Score Format Package.

3.3.1 Package Metadata Files

The following files **MUST** be present:

- `META-INF/container.xml`: A description of the *default rendition*, and any *alternate renditions* of the content available in the package. It describes the root location and the Internet Media (MIME) Type [4] of each rendition. This file **MUST NOT** be encrypted.
- `META-INF/metadata.xml`: A XML file containing the catalogue metadata for the package and default rendition. Refer to §5.

The following files **MAY** be present:

- `META-INF/manifest.xml`: A description of all files in package including their size, Internet Media (MIME) type and **OPTIONAL** information related to the digital signature of the package

  A manifest **MUST** be present if any of the following conditions is met:

  - Any file in the package is digitally signed.
  - Any particular rendition makes use of more than one media object – e.g. one that is not referenced from the `META-INF/container.xml` file.

- One or more folders containing files associated with alternate renditions.

3.3.2 Use of ZIP

The Open Score Format supports a subset of the ZIP format as described in Zip File Format Specification [8]:

- Multi-volume ZIP files **MUST NOT** be used.
- Encryption mechanisms provided by the ZIP format **MUST NOT** be used; instead mechanisms provided in Open Score Format Security Model Specification [9] **MAY** be used.
- File system names within an OSF Package **MUST** be encoded in UTF8.
- The ZIP64 extensions **MAY** be used and **MUST** be supported by compliant Open Score Format reading applications.
- The ZIP compression method **MAY** be either STORE (0) or DEFLATE (8). Other compression methods **MUST NOT** be used.[3]

Compliant Open Score Format reading applications **MUST** reject as invalid ZIP files using the following ZIP features:

- Archive Decryption Headers or Archive Extra Data Headers
- Multiple-volume archives
- Encryption Features
- Compression methods other than 0 (STORE) and 8 (DEFLATE)

3.4 Use of XML

XML1.0 is used throughout the Open Score Format:

- The package's `container.xml`, `metadata.xml` and `manifest.xml` files
- Package metadata
- Score (MusicXML)

A W3C XML Schema is provided for each of these document types.

All XML files within an Open Score Format package **MUST**:

- Be well-formed
- Contain a processing instruction that references the appropriate schema
- Validate against the appropriate schema

3.5 Open Score Format Internet Media (Formerly MIME) Type Identifiers

The OSF defines the following Internet Media Types [4].

Media Type	Extension	Description
`application/osf-package`[4]	`osf`	Open Score Format package file
`application/osf-score-pvg-profile+xml`	`osfpvg`	PVG profile Music XML Score

[3] Other compression algorithms are supported by the ZIP standard, but support for these is not universal. The STORE and DEFLATE are known to be supported by Java, .NET and zlib.

[4] Provisional, and unregistered internet media types. Final values to be confirmed.

3.6 Use of Internationalized Resource Identifiers (IRIs)

References from one file to another in an Open Score Format package are encoded using relative Internationalized Resource Identifiers (IRIs). [7].

The following restrictions are placed on IRIs used within the package:

- References between files contained within a package **MUST**:
    - Be relative.
    - Not reference files outside of the root of the package.
- Absolute references to resources external to the package **MAY** be used with the exception of file:// IRIs which **MUST NOT** be used. This rule is enforced by the schemas for the `container.xml` and `manifest.xml` files.[5]

3.7 `META-INF/container.xml` file

The `META-INF/container.xml` file describes the renditions of the content available in the package. The W3C XML Schema for this document can be found in Section 7.1.

One of the renditions is the *default rendition*. It **MUST** refer to a MusicXML file that is compatible with one of the Open Score Format Digital Score Application Profiles, and thus usable by any Open Score Format compliant application.

```
<container xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="http://www.musicxml.org/xsd/container.xsd">
    <rootfiles>
          <rootfile full-path="Default/Score.xml"
                    media-type="application/osf-pvg-application-profile"/>
          <rootfile full-path="Enhancement/Enhancement.xml"
                    media-type="applicatication/myAppScoreEnhancement"/>
    </rootfiles>
</container>
```

Figure 2: Example of the description of the packages renditions

Each rendition is specified by a `rootfile` element that has:

- A `full-path` attribute that contains an IRI that specifies relative to the package root the location of root file of the rendition. In the case of the default rendition this will be a MusicXML file.
- A `media-type` attribute that indicates the content type of the rendition

The first `rootfile` element declared is the *default rendition.* The `container.xml` file **MUST NOT** be encrypted.

3.8 `META-INF/manifest.xml` file

The `META-INF/manifest.xml` describes constituent parts of the package. It contains:

- A description of each asset contained in the package
- Descriptions of *asset-groups* (asset groups are **OPTIONAL** logical groupings of assets)
- Digital signature data (**OPTIONAL**)
- An X.509 public key certificate of the package signer (**OPTIONAL**).

[5] Implementers of applications that process Open Score Format packages must take extreme care to avoid security hazards associated with fetching documents' raw IRIs. It is recommended that an explicit check for disallowed file:// IRIs is included.

The manifest **MAY** be omitted for simple packages where:

- All content is referenced by the `container.xml`, and;
- Package signing is not used

The procedure to sign packages and the signature data included in the manifest is described in §11

3.8.1 *Declaration of Assets and Asset-Groups*

Asset and asset-groups are declared with `asset` and `asset-group` elements respectively. These are children of an `assets` element.

```
<osfm:assets name="ExampleManifest">
    <osfm:asset name="META-INF/Container.xml" media-type="text/xml"
                signed="true" c14n="true"/>
    <osfm:asset name="META-INF/Manifest.xml" media-type="text/xml"
                signed="true" c14n="true"/>
    <osfm:asset name="META-INF/Metadata.xml" media-type="text/xml"
                signed="true" c14n="true"/>
    <!-- default rendition -->
    <osfm:asset name="Default/Score.xml"
                media-type="application/application-osf-score-pvg-profile"
                signed="true" c14n="true"/>
    <!-- altnernate rendition -->
    <osfm:asset name="Alternate/Enhancement.xml"
                media-type="applicatication/myAppScoreEnhancement"
                signed="true" c14n="true"/>
    <osfm:asset name="Image1.png" media-type="image/png" signed="true" c14n="false"/>
    <osfm:asset name="Image2.png" media-type="image/png" signed="true" c14n="false"/>
    <osfm:asset name="Image3.png" media-type="image/png" signed="true" c14n="false"/>
    <osfm:asset name="Image4.png" media-type="image/png" signed="true" c14n="false"/>
    <osfm:asset-group name="AlternateRendition">
        <osfm:asset-reference ref="Alternate/Enhancement.xml"/>
        <osfm:asset-reference ref="Image1.png"/>
        <osfm:asset-reference ref="Image2.png"/>
        <osfm:asset-reference ref="Image3.png"/>
        <osfm:asset-reference ref="Image4.png"/>
    </osfm:asset-group>
</osfm:assets>
```

Figure 3: Example of an asset and asset-group declaration

Each asset in the package (including the package metadata files `container.xml` and `manifest.xml` and `metadata.xml`), **MUST** be declared with an **`asset`** element:

- A `size` attribute that specified the extent of the asset in bytes. (**OPTIONAL)**[6]
- A `name` attribute that contains an IRI indicating the location of the file relative to this one.
- A `mediaType` attribute that indicates the content type of the file.

[6] The size element is provided to support encryption of assets (the mechanisms for doing so are not defined here). In some cases where a block cipher is used, the encrypted file included in the package is larger than the original. The size element allows the original file-size to be recovered.

- A `signed` attribute indicates whether the asset should be included in the digital signature of the package. (**OPTIONAL**)
- A `c14n` attribute indicates whether the asset should be canonicalized during digital signature processing. **(OPTIONAL).** Refer to §4.3.2.

Assets **CANNOT** be declared more than once.

An `asset-group` element **MAY** be used to declare logical relationships between assets. The `name` attribute specifies the unique identifier for the grouping. The Open Score Format attaches no significance to any particular identifier value, but applications that make use of alternate renditions **MAY** do so.

The assets within the `asset-group` element are declared using an `asset-reference` child-element. The `ref` attribute contains the IRI of an asset, which **MUST** previously have been declared with an `asset` element.

4. Package Signing

4.1 Introduction

Package signing uses a combination of message digests and digital signatures. It is used to:

- Maintain integrity of the *assets* and metadata in the package.
- Verify the authenticity of the package and its creator.

Both the message digests for assets and the package signature are included in the `META-INF/manifest.xml` file.

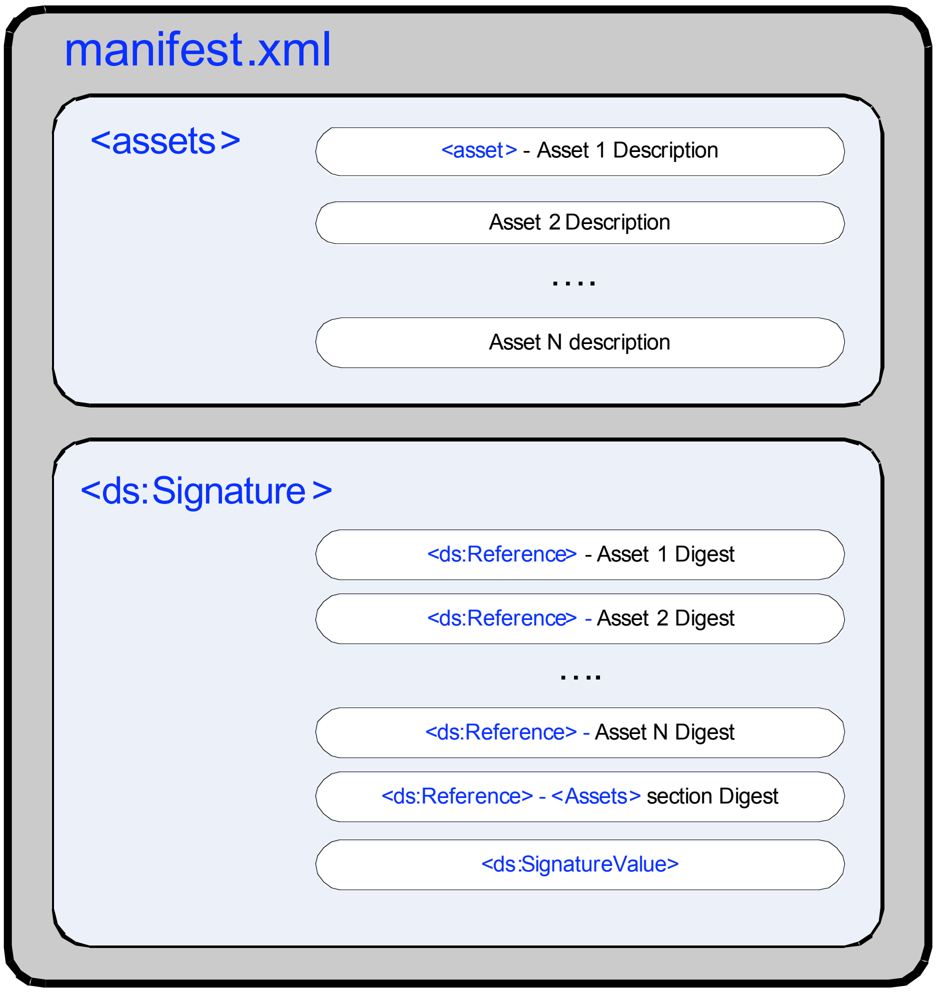
manifest.xml
<assets>
<asset> - Asset 1 Description
Asset 2 Description
....
Asset N description
<ds:Signature>
<ds:Reference> - Asset 1 Digest
<ds:Reference> - Asset 2 Digest
....
<ds:Reference> - Asset N Digest
<ds:Reference> - <Assets> section Digest
<ds:SignatureValue>

Figure 4: Structure of a signed `META-INF/manifest.xml` filee

The algorithms and XML data formats for digital signatures specified by *The XML Signature Syntax and Processing Specification (Second Edition)* (XML-SIG) [9] are used.

Package signing consists of the following steps:

1. Calculating message digests of each asset, including `META-INF/manifest.xml`
2. Calculating and enveloping signature over the resulting message digests

The combination of these two steps allows tampering with the package assets and metadata to be detected.

An example of a `META-INF/manifest.xml` file containing signatures can be found in §7.4

4.2 Keys Used for Signing

The creator or publisher of a package signs it at the time of creation or modification, prior to distribution to the customer.

The party signing the package requires a public-key encryption key-pair, the public key of which is signed by a certificate authority in the form of an X.509 public-key certificate.

The X.509 public-key certificate is publicly distributed, optionally in the package itself.

4.3 Inclusion of Signature Data in `META-INF/manifest.xml`

Package integrity metadata are included in the `META-INF/manifest.xml` file and consist of message digests of package assets, and a signature of the manifest itself.

Section 3.8 describes the purpose and contents of the of the non-signature data in the manifest. The remainder of the data consists of a XML-SIG `ds:Signature` element containing all message digests and signatures.

It should be noted that the XML-DSIG specification allows wide choice of algorithms in the signing process. An OSF-reading application **MUST** expect any combination permitted by XML-DSIG.

The following sections describe one possible choice of algorithms in the signing process.[7]

4.3.1 Inclusion of Message Digests Values for Assets Under a `ds:Signature` Element

A message digest **MAY** be included in the manifest for each asset in an Open Score Format package. The digest allows tampering with the asset to be detected.

It is **RECOMMENDED** that any asset that has been encrypted is also protected with a message digest. If the asset is encrypted, the digest **MUST** be calculated over the encrypted rather than unencrypted form of the asset.[8]

The message digest of each asset is described with a `ds:Reference` element.

```
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="../META-INF/Container.xml">
            <ds:Transforms>
                <ds:Transform
              Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-ds:Signature"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>2a3Xy5hSLQMumef8YbNwjo/eIEY=</ds:DigestValue>
        </ds:Reference>
```

Figure 5: Fragment of `META-INF/manifest.xml` showing message digest for an XML document using canonicalization.

The URI element of the `ds:Reference` element contains a IRI that references a package asset. Refer to §3.6 for further details.

The message digest algorithm used is specified with the `Algorithm` attribute of the `ds:DigestMethod` element. Any algorithm supported by XML-SIG may be used. A value of http://www.w3.org/2000/09/xmldsig#sha1 is **RECOMMENDED**.

[7] The examples here were generated with the Open Score Format Packaging Toolkit which is implemented using the Microsoft .NET implementation of XML-DSIG

[8] An application that is verifying the signature of the package may not be capable of decrypting the asset.

4.3.2 Specifying the Signature and Canonicalization Algorithm to be Applied to a `ds:Signature` Element

XML-SIG allows a choice of digital signature and canonicalization algorithms.

XML *assets* can optionally be canonicalized before signing. The process of canonicalization (C14N) attempts to normalize the content of the document whilst ignoring non-content differences such as white-space or comments.

It is **RECOMMENDED** that all XML *assets* are canonicalized prior to signing. Note that canonicalization cannot compensate for differences in element or attribute ordering in documents.[9]

The choice of algorithms is specified by the `ds:CanonicalizationMethod` and `ds:SignatureMethod`:

```
<ds:Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
          <ds:CanonicalizationMethod
                  Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
...
...
```

Figure 6: Fragment of the `META-INF/Manifest.xml` file showing specification of canonicalization and signature algorithms

Any algorithm supported by XML-SIG **MAY** be used. The table below shows those that are **RECOMMENDED**:

Name	Recommended Algorithm URI
Canonical XML Version 1.0	`http://www.w3.org/TR/2001/REC-xml-c14n-20010315`
RSA-SHA1 - PKCS1- Digital Signature Algorithm	`http://www.w3.org/2000/09/xmldsig#rsa-sha1`

Table 3: Recommended algorithms for digital signatures

4.3.3 Inclusion of a Message Digest for `META-INF/manifest.xml`

A message digest is calculated `META-INF/manifest.xml` in the same manner as other assets, with the `URI` property of the `ds:Reference` element containing a null path.

The Enveloping Signature Transform must be applied to this operation to exclude the contents of the `ds:Signature` element from the digest.

```
<ds:Reference URI="">
    <ds:Transforms>
        <ds:Transform
            Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-ds:Signature"/>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <ds:DigestValue>6jHiD24vvcpBsVyrrQMQgaSs1Zs=</ds:DigestValue>
</ds:Reference>
```

[9] This is a particular concern for MusicXML documents as there are many ways of notating a measure of multi-voice music with <`backup`> and <`forward`> elements which, whilst conveying the same semantics, result in different content models after canonicalization

4.3.4 *Signature Value*

The signature value is specified using a `ds:SignatureValue` element. The bytes of the signature are Base-64 encoded.

```
<ds:Signature>
    ...

<ds:SignatureValue>KgqGUt7U97Kg4QinaB7KgiL7kBzFnsgLfRbHpQNhrvFpwkUuIq/gQ75klgLNV4n/S
                  X9PV5fi7l5RW/sA/p32FAyaPNU2GYWwjTWn53jMwt1maPFKKsWAN4g9W4NPkABZaU
                  XHZ6KQnAyFo6wEJ6wXaAHKxLbFyxPSKycoB/lLRk4=
</ds:SignatureValue>
    ...
</ds:Signature>
```

Figure 7: Fragment from the `META-INF/manifest.xml` file showing a digital signature value

4.4 Including a X.509 Certificate in `META-INF/manifest.xml`

An X.509 certificate bearing the signer's public key **MAY** be included in the `META-INF/Manifest.xml` file. Doing so is **RECOMMENDED** when a package is signed, although other means of key distribution are possible.

The X.509 certificate is base-64 encoded inside a XML-SIG `ds:X509Certificate` element. Whilst the XML-SIG `ds:KeyInfo` element supports other key data formats, only `ds:X509Data` **SHALL** be used.

Similarly, the `ds:X509Data` element allows beneath it allows various elements that can describe a certificate (or where one might be obtained). Only `ds:X509Certificate` **SHALL** be supported.

```
<ds:Signature Id="ManifestSignature">
    <ds:SignedInfo>
        ...
     ...
         <ds:KeyInfo>
               <ds:X509Data>
                     <ds:X509Certificate>
                                 Q2VydGlma..........ICA2ODo5Zg==
               </ds:X509Certificate>
         </ds:X509Data>
    </ds:KeyInfo>
</ds:Signature>
```

Figure 8: Fragment of `META-INF/Manifest.xml` showing the inclusion of an X.509 certificate

4.5 Sequence of Operations for Signing a Package

1. Determine which of the assets require protecting by the digital signature (usually the union of all `asset` elements in each `assets-group` whose `signed` attribute is `true`, and `META-INF/metadata.xml`, `META-INF/container.xml`).
2. Calculate the message digest for each asset identified in step 1. Populate the `ds:Signature/ds:SignedInfo` element in `META-INFO/manifest.xml` with `ds:Reference` elements describing the assets and their digests.
3. Calculate the message digest for `META-INFO/manifest.xml` using the enveloping signature transform [see §6.6.4 XML-SIG]
4. Add a `ds:KeyInfo` element to `META-INFO/manifest.xml` to hold the X.509 certificate that contains the public key of the signer [**OPTIONAL**]

5. Apply the digital signature algorithm (and C14N transformation [§6.5.1 XML-SIG] to the `ds:SignedInfo` element and all of its contents.
6. Create a `ds:SignatureValue` element containing the signature value and add it as a child of `ds:Signature`.

This section builds on §3.1 (Core Generation) of the XML-SIG specification [9]. It is likely that a library implementation of XML-SIG will automate most of these steps.

4.6 Validating the Signature on a Package

1. Obtain the X.509 certificate for the signer of the package. If it has been included in the package it can be extracted from `META-INF/metadata.xml` with the XPATH expression: `osfm:manifest/ds:Signature/ds:KeyInfo/ds:X509Certificate`
2. Recover the digital signature value from the `ds:SignatureValue` element.
3. Validate the signature on the `ds:SignedInfo` element, applying the C14N transformation first.
4. Calculate the message digest for each asset included in a `ds:Reference` element of the signed manifest. Compare the calculated digest with that included in the signature.

This section builds on §3.1 (Core Generation) of the XML-SIG specification [9]. It is likely that a library implementation of XML-SIG will automate most of these steps.

5. Metadata

5.1 Introduction

The Open Score Format specifies a minimum requirement for package catalogue metadata. This is called the *Open Score Format Base-line metadata profile*.

The purpose of the catalogue metadata is to describe the package – its author, title, publisher and so on. This data may be used in applications that use Open Score Format packages such as e-commerce or archive systems to build a catalogue.

Rather than develop a complete application specific metadata schema, the Open Score Format specification has adopted existing standards. In particular the *Qualified Dublin Core* from the Dublin Core Metadata Initiative (DMCI) [2] provides a metadata vocabulary and encoding schemes for much of the catalogue metadata.

This has been augmented by a vocabulary specific to the music publishing domain that enhances the description of the score, including meter, key and opus number.

5.2 Open Score Format Base-line Metadata Profile

The *Open Score Format Base-line Metadata Profile* specifies the required, recommended and optional metadata terms that must be present in the `META-INF/metadata.xml` file, and the way in which they must be encoded.

A W3C XML Schema is provided that can be used to validate metadata document instances. Due to the manner in which the Qualified Dublin Core is designed, the schema cannot enforce all requirements of the *Open Score Format Base-line metadata profile*; the Open Score Format package validation tool does however provide this higher-level validation. Refer to §6 for further details.

Open Score Format applications **SHOULD** at a minimum validate `META-INF/metadata.xml` files against the schema.

5.3 Design Objectives

The design of the metadata representation in Open Score Format aims to strike a balance between providing a rigid definition and one that is fully flexible.

Using the Dublin Core vocabulary rather than defining another one specific to this application is beneficial in that the Dublin Core is already widely employed as the data format for catalogue metadata in digital and physical goods. Using the same vocabulary and encoding schemes allows straightforward interoperability of metadata.

By default, the Dublin Core provides a highly permissive definition of metadata – both in terms of structure and encoding schemes. This is advantageous to those applying the Dublin Core to a wide range of digital resources, but disadvantageous to software developers who develop tools that process the resulting data.

To avoid undue complexity in tools that read or write Open Score Format packages, the *Open Score Format Base-line Metadata Profile* imposes additional constraints on the cardinality of Dublin Core terms, and their encoding schemes. Furthermore, a set of recommendations is provided below for mapping of Dublin Core metadata vocabulary into the music-publishing domain.

Consideration was given to using the Resource Description Framework [5] in conjunction with Dublin Core. This was rejected on grounds of complexity. The design of the *Open Score Format Base-line metadata profile* has been undertaken with the consideration that applications might want to transcode to and from RDF triples. Implementing transcoding into RDF triples is believed to be straightforward.

5.4 The `META-INF/metadata.xml` file

The `META-INF/metadata.xml` contains metadata for the package and *default rendition*. The file contains an `osf-package-metadata-baseline` root element containing a number of child elements:

- Elements defined by the Dublin Core (the Dublin Core metadata 'terms'). Refer to §5.5 for further details.
- Elements defined in the `osfmeta` namespace that define metadata specific to the music publishing domain. Refer to §5.8 for further details.
- An `osf-meta-app-ext-metadata` element for adding application specific metadata. Refer to §5.9 for further details.

```xml
<osfmeta:osf-package-metadata-baseline xmlns:dcterms="http://purl.org/dc/terms/"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns="http://openscoreformat.sourceforge.net/osf/metadata"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:osfmeta="http://openscoreformat.sourceforge.net/osf/metadata"
        xsi:schemaLocation="http://openscoreformat.sourceforge.net/osf/metadata
                            http://openscoreformat.sourceforge.net/1.0/metadata.xsd">

    <dc:title xml:lang="fr">Marche Slave</dc:title>
    <dc:title xml:lang="de">Marsche Slawonisch</dc:title>
    <dc:title xml:lang="en">Slavonic March</dc:title>
    <dcterms:alternative>A slvanic march</dcterms:alternative>
    <dc:creator>Pyotr Ilyich Tchaikovsky</dc:creator>
    <dc:description>Piano Solo</dc:description>
    <dc:publisher>Music Sales</dc:publisher>
    <dcterms:created xsi:type="dcterms:W3CDTF">1876</dcterms:created>
    <dcterms:issued xsi:type="dcterms:W3CDTF">2008-03-12</dcterms:issued>
    <dcterms:modified xsi:type="dcterms:W3CDTF">2008-03-12</dcterms:modified>
    <dc:type xsi:type="dcterms:DCMIType">InteractiveResource</dc:type>
    <dcterms:medium xsi:type="dcterms:IMT">osfpvg</dcterms:medium>
    <dc:identifier xsi:type="dcterms:URI">
          http://openscoreformat.sourceforge.net/OSF/Content/0123456789
    </dc:identifier>
    <dcterms:source xsi:type="dcterms:URI">
          http://openscoreformat.sourceforge.net/
    </dcterms:source>
    <dc:language xsi:type="dcterms:RFC4646">en</dc:language>
    <dc:rights>(C) Some company 2008</dc:rights>
    <osfmeta:meter>
          <osfmeta:beats>4</osfmeta:beats>
          <osfmeta:beat-type>4</osfmeta:beat-type>
    </osfmeta:meter>
    <osfmeta:opus>31</osfmeta:opus>
    <osfmeta:key>
          <root>B</root>
          <accidental>flat</accidental>
          <mode>minor</mode>
    </osfmeta:key>
</osfmeta:osf-package-metadata-baseline>
```

Figure 9: Example `META-INF/metadata.xml` file

5.5 Metadata Encodings

5.5.1 Introduction

This section describes the way in which metadata is encoded using the Dublin Core elements, and provides recommendations for the way in which these map onto the concepts and terminology of the music publishing domain. These recommendations are based on those in the *Dublin Core User Guide* [20].

5.5.2 Specifying Element Encodings

The *Open Score Format Base-line Metadata Profile* enforces the use of content encodings for Dublin Core elements. This is a restriction of the content of the inner-XML of the element.

Refer to Table 6: Use of Qualified Dublin Core elements for a complete description of encodings for particular Dublin Core elements.

In a metadata instance document, the encoding type is specified by using an `xsi:type` attribute on any Dublin Core element:

```
<dc:identifier xsi:type="dcterms:URI">
http://www.yamaha.com/OSF/Content/0123456789
</dc:identifier>
```

Figure 10: Use of a `dc:identifier` element

The use of encodings is not enforceable by the *Open Score Format Base-line Metadata Profile* schema. Use the Open Score Format validation tool instead.

5.5.3 Uniform Resource Identifiers

Uniform Resource Identifiers [7] (URIs) are often used in metadata fields that declare or reference a unique resource (for instance single item of content or a content delivery or rendering platform).

Any valid and unique URI may be used. It is the responsibility of the content publishers to allocate unique identifiers and chose an encoding scheme.

The following encoding schemes are **RECOMMENDED**:

- Uniform Resource Locators (URLs)
- Uniform Resource Names (URNs)

5.5.3.1 Uniform Resource Locators

A uniform resource locator (URL) indicates the location at which a resource can be found and retrieved.

There is no strict requirement that a URL used to describe a resource is retrievable from the location indicated – only that the URL is unique - although in the case of a score download service being able to retrieve the content (or redirecting to a location from which it is possible) is beneficial.

5.5.3.2 Uniform Resource Names

Uniform Resource Names [15] (URNs) are a sub-class of URIs intended to be used to encode location independent, persistent, resource identifiers.

Further refinements to the encoding scheme exist for encoding classes of existing identifiers such as ISBNs [16] and Universally Unique Identifiers[10] (UUIDs) [17].

The use of ISBNs may be of interest to publishers who also publish physical goods and chose to also allocate ISBNs to their electronic goods. They can also be used to reference physical goods carrying an ISBN.

[10] Globally Unique Identifiers (GUIDs) are an implementation of UUIDs and can be used interchangeably.

5.5.3.3 Example Identifiers and Encodings

Identifier	Encoding	
ISBN: `978-0307266934`	URN-ISBN	`URN:ISBN:978-0307266934`
UUID: `67BB75A1-964E-4073-989F-E68DBEE84D68`	URN-UUID	`URN:UUID:67BB75A1-964E-4073-989F-E68DBEE84D68`
Arbitrary identifier: `1234567890`	URL	`http://www.ascoredownloadservice.com/openscoreformat/content/1234567890`

Table 4: Table showing the ways in which a number of different identifier classes can be encoded as a URI

5.5.4 W3C Time and Date Format

The W3C Time and Date Format allows time to be specified at varying levels of precision. Further details can be found in Date and Time Formats [18].

Precision	**Example**
Year	`1997`
Year and month	`1997-07`
Complete date	`1997-07-16`
Complete date plus hours and minutes	`1997-07-16T19:20+01:00`
Complete date plus hours, minutes and seconds	`1997-07-16T19:20:30+01:00`
Complete date plus hours, minutes, seconds and decimal fraction of a second	`1997-07-16T19:20:30.45+01:00`

Figure 11: Examples of times and dates at different degrees of precision

5.5.5 DCMI Period

The DCMI period is an encoding scheme for a time interval. Refer to [19] for further details.

5.6 Language Variants

Elements without a content encoding specified have the type `xs:string`. The language of content of the element can be specified using an `xml:lang` attribute with a RFC4646 [14] language identifier.

Most Dublin Core elements can appear multiple times in an instance document to support language variants:

```
<dc:title xml:lang="fr">Marche Slave</dc:title>
<dc:title xml:lang="de">Marcshe Slawonisch</dc:title>
<dc:title xml:lang="en">Slavonic March</dc:title>
```

Figure 12: Example of repeated Dublin core elements used to support language variants.

5.7 Recommended use of Dublin Core Metadata Terms

5.7.1 `dc:title`

Describes the title of the content. May appear multiple times to support language variants.

Alternative titles should be encoded using the `dcterms:alternative` element instead.

5.7.2 *`dc:creator`*

Describes the creator(s) of the content. This may include the name of a person, organisation or service. Multiple elements may appear to describe:

- Composer(s)
- Transcribers(s)
- Arranger(s)

The order of multiple elements may be used to imply precedence. It is **RECOMMENDED** that one `dc:creator` element appears per creator.

Personal names should appear with the family name first followed by given name:

```
<dc:creator>Pyotr Ilyich Tchaikovsky</dc:creator>
```

Figure 13: Recommended encoding of a personal name.

5.7.3 *`dc:description`*

A description of the content. This element may be used for a number of purposes:

- The type of work: e.g. Piano Solo, Song, String Quartet, Concerto etc.
- The structure of the work: e.g. sonata form, fugue, march, song-form, variation and theme etc.
- Intended purpose of work: e.g. religious work, military march, dance etc.

Where a work is comprised as several distinct sections, a `dcterms:tableOfContents` element should be used to describe each one.

5.7.4 *`dc:publisher`*

The name of the publisher.

5.7.5 *`dc:created`*

The date at which the package was first created in W3C Time and Date format.

5.7.6 *`dc:issued`* *(**OPTIONAL**)*

The date at which the package was formally issued – perhaps before which release is embargoed.

5.7.7 *`dc:modified`*

The date at which the package was last modified.

5.7.8 *`dc:valid`*

The date (or range of dates) during which the package is valid.

5.7.9 *`dc:dateCopyrighted`*

The copyright statement date.

5.7.10 *`dc:type`*

The type of the work encoded using the DCMI Type vocabulary. A value of `InteractiveResource` is recommended:

```
<dc:type xsi:type="dcterms:DCMIType">InteractiveResource</dc:type>
```

Figure 14: Use of the `dc:type` element

5.7.11 *`dcterms:medium`*

The Internet Media Type (MIME type) of the work – e.g. `osfpvg` or `osf`.

5.7.12 *dc:identifier*

A unique identifier for the content encoded as a URI. See §5.5.3.

5.7.13 *dc:source*

An identifier of a resource from which the work is derived. For example, the original edition if this one is an arrangement.

5.7.14 *dc:language*

The language(s) of the content, encoded using RFC1766 [14]. Use multiple `dc:language` elements if multiple language variants are provided by the content.

5.7.15 *dc:relation*

Describes the relationship of the resource to others. The Qualified Dublin Core provides qualified terms that allow more precise relationships to be described. These **SHOULD** be used in preference to `dc:relation`.

All are encoded as URIs.

Qualified Dublin Core Relation Term	Use
`dc:isVersionOf`	Resource is a version of another by the same creator.
`dc:hasVersion`	Resource has other versions by the same creator.
`dc:isReplacedBy`	Resource is superseded by another.
`dc:replaces`	Resource is a replacement for another.
`dc:isRequiredBy`	Resource is required by another in order for it to be useful.
`dc:requires`	Resource requires another resource to be useful.
`dc:isBasedOn`	Resource is derivative work of another, such as an arrangement.
`dc:isBasisOf`	Resource has derivative works, e.g. arrangements.

Table 5: Qualified Dublin Core Terms that can be used in place of `dc:relation`.

5.7.16 *dc:rights*

Contains a rights management statement for the resource. This could include a URL that refers to a license agreement or a statement that a digital rights management in force.

5.8 Use of OSF Metadata Terms

5.8.1 *osfmeta:meter*

Used to describe meter(s) or time signature(s) of the resource.When multiple significant meters are present in a work, multiple `osfmeta:meter` elements may appear in order of significance (rather than occurrence).

```
<osfmeta:meter>
        <osfmeta:beats>4</osfmeta:beats>
    <osfmeta:beat-type>4</osfmeta:beat-type>
</osfmeta:meter>
```

Figure 15: Example of `osfmeta:meter` element to describe the meter(s) of a resource.

5.8.2 `osfmeta:key`

Used to describe the significant key signature of the resource. When the resource has multiple significant key signatures, multiple `osfmeta:key` elements may appear in order of significance (rather than occurrence).

```
<osfmeta:key>
    <osfmeta:root>B</osfmeta:root>
    <osfmeta:accidental>flat</osfmeta:accidental>
    <osfmeta:mode>minor</osfmeta:mode>
</osfmeta:key>
```

Figure 16: Use of `osfmeta:key` element to describe key signature(s) or a resource.

Possible values for `osfmeta:key` are `A`, `B`, `C`, `D`, `E`, `F`, `G` and `none`[11].

Possible values for the `osfmeta:accidental` element are: `sharp` and `flat`. The element may be omitted for natural keys.

Possible values for the `osfmeta:mode` element are: `minor`, `major`, `dorian`, `phrygian`, `lydian`, `mixolydian`, `aeolian`, `ionian`, `locrian`, `none`[12]

5.8.3 `osfmeta:work-number`

Used to describe the work (e.g. opus number) of a resource, if present.

5.9 Addition of Application-specific Metadata to `META-INF/metadata.xml`

Application specification metadata can be incorporated into `META-INF/metadata.xml` using the substitution group mechanism provided by W3C XML Schema.

The schema for `metadata.xml` provides an empty placeholder element - `osfmeta:app-ext-metadata` - under the `osf-package-metadata-baseline` element, that may appear multiple times in an instance document.

An application can use a W3C XML Schema to define complex types that represent their data with the `substitutionGroup="osfmeta:app-ext-metadata"` attribute. These can be used instead of the `osfmeta:app-ext-metadata` placeholder element in an instance document.

5.9.1 *Example of an application-specific data schema*

```
<?xml version="1.0" encoding="UTF-8"?>
<osfmeta:osf-package-metadata-baseline
        xmlns:dcterms="http://purl.org/dc/terms/"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns="http://openscoreformat.sourceforge.net/osf/metadata"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:osfmeta="http://openscoreformat.sourceforge.net/osf/metadata"
        xmlns:exampleApp="http://openscoreformat.sourceforge.net/exampleApp/"
        xsi:schemaLocation="http://openscoreformat.sourceforge.net/exampleApp/
           http://openscoreformat.sourceforge.net/1.0/exampleDerivedSchema.xsd">

        <dc:title xml:lang="fr">Marche Slave</dc:title>
    ...

    <exampleApp:exampleAppExtension>
          <exampleApp:appData1>foo</exampleApp:appData1>
```

[11] Used for scores in unconventional keys, those that are atonal or where the composer or engraver has chosen to omit the key-signature and use accidentals throughout

[12] Used for scores without an obvious mode.

```
          <exampleApp:appData2>bar</exampleApp:appData2>
          <exampleApp:appData3>barred</exampleApp:appData3>
    </exampleApp:exampleAppExtension>
```
Figure 17: Example of a schema for defining application-specific data extensions.

```
<?xml version="1.0" encoding="UTF-8"?>
<osfmeta:osf-package-metadata-baseline
       xmlns:dcterms="http://purl.org/dc/terms/"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns="http://openscoreformat.sourceforge.net/osf/metadata"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:osfmeta="http://openscoreformat.sourceforge.net/osf/metadata"
    xmlns:exampleApp="http://openscoreformat.sourceforge.net/exampleApp/"
    xsi:schemaLocation="http://openscoreformat.sourceforge.net/exampleApp/
          http://openscoreformat.sourceforge.net/1.0/exampleDerivedSchema.xsd">
    <dc:title xml:lang="fr">Marche Slave</dc:title>
    ...

    <exampleApp:exampleAppExtension>
          <exampleApp:appData1>foo</exampleApp:appData1>
          <exampleApp:appData2>bar</exampleApp:appData2>
          <exampleApp:appData3>barred</exampleApp:appData3>
    </exampleApp:exampleAppExtension>
```
Figure 18: Example of `META-INF/metadata.xml` containing application-specific data.

6. Open Score Format Packaging Tool

The Open Score Format Packaging Tool can be used to check validity of an Open Score Format Package. The checks are relevant to packaging and metadata:

- Validating all XML files against their respective schemas.
- Ensuring that all referenced files exist.
- Ensuring that incompatible features of the ZIP compression format are not used (See §3.3.2).
- Enforcing use of content encodings for metadata elements (See §5.5.2).

  Documentation for the tool can be found on the OpenScoreFormat Project Site at http://openscoreformat.sourceforge.net/

7. Appendix

7.1 META-INF/container.xml W3C XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefaul
    <xs:annotation>
          <xs:documentation>Open Score Format

Version 1.0 – 11 September 2009

Copyright © 2004-2009 Recordare LLC.
http://www.recordare.com/

This MusicXML™ work is being provided by the copyright holder under the MusicXML Document Type
Definition Public License Version 2.0, available from:

    http://www.recordare.com/dtds/license.html

Starting with Version 2.0, the MusicXML format includes a standard zip compressed version. These
zip files can contain multiple MusicXML files as well as other media files for images and sound.
The container schema describes the contents of the META-INF/container.xml file. The container describes
the starting point for the MusicXML version of the file, as well as alternate renditions such as PDF and
audio versions of the musical score.

The MusicXML 2.0 zip file format is compatible with the zip format used by the java.util.zip package and
Java JAR files. It is based on the Info-ZIP format described at:

    ftp://ftp.uu.net/pub/archiving/zip/doc/appnote-970311-iz.zip

The JAR file format is specified at:

    http://java.sun.com/javase/6/docs/technotes/guides/jar/jar.html

Note that, compatible with JAR files, file names should be encoded in UTF-8 format.
```

```
Files with the zip container are compressed the DEFLATE algorithm. The DEFLATE Compressed Data Format
(RFC 1951) is specified at:

    http://www.ietf.org/rfc/rfc1951.txt

The recommended file suffix for OSF files is .osf. The recommended media type for an Open Score Format
package file is:

          application/vnd.yamaha.openscoreformat

The recommended media type for an OSF PVG profile MusicXML score is:

          application/vnd.yamaha.openscoreformat.osfpvg+xml

Open Score Format specifies additional constraints on container files than does MusicXML 2.0. See the
Open Score Format Packaging Specification for further details.

Open Score Format uses XSD for all its XML format definitions. This container.xsd file is an XSD version
of MusicXML 2.0's container.dtd file.
  </xs:documentation>
    </xs:annotation>
    <xs:element name="container">
          <xs:annotation>
                <xs:documentation>Container is the document element. </xs:documentation>
          </xs:annotation>
          <xs:complexType>
                <xs:sequence>
                      <xs:element name="rootfiles">
                            <xs:annotation>
                                  <xs:documentation>
            Rootfiles include the starting points for the different representations of a
            MusicXML score. The MusicXML root must be described in the first rootfile element.
            Additional rootfile elements can describe alternate versions such as PDF and audio files.
            </xs:documentation>
                            </xs:annotation>
```

```
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="rootfile" maxOccurs="unbounded">
                                          <xs:annotation>
                                                <xs:documentation>
                  The rootfile element describes each top-level file in the MusicXML container.
                  </xs:documentation>
                                          </xs:annotation>
                                          <xs:complexType>
                                                <xs:attribute name="full-path" type="xs:anyURI">
                                                      <xs:annotation>
                                                            <xs:documentation>
                      The full-path attribute provides the path relative to the root folder of the
                      zip file. It is an IRI as defined in RFC3987.</xs:documentation>
                                                      </xs:annotation>
                                                </xs:attribute>
                                                <xs:attribute name="media-type" type="xs:anyURI">
                                                      <xs:annotation>
                                                            <xs:documentation>
                      The media-type identifies the type of different top-level root files. It is an
                      Internet Media Type (MIME-Type) as defined by RFC2046. It is an error to have a
                      non-MusicXML media-type value in the first rootfile in a rootfiles element. If no
                      media-type value is present, a MusicXML file is assumed. A MusicXML file used as a
                      rootfile may have score-partwise, score-timewise, or opus as its document element.
                      </xs:documentation>
                                                      </xs:annotation>
                                                </xs:attribute>
                                          </xs:complexType>
                                    </xs:element>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
</xs:element>
```

```
</xs:schema>
```

7.2 META-INF/manifest.xml W3C XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
* Version: 1.0, 3/7/09
*
* Copyright © 2009 Yamaha Corporation
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above copyright
*       notice, this list of conditions and the following disclaimer in the
*       documentation and/or other materials provided with the distribution.
*     * Neither the name of the <organization> nor the
*       names of its contributors may be used to endorse or promote products
*       derived from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY YAMAHA CORPORATION ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL YAMAHA CORPORATION BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:osfm="http://openscoreformat.sourceforge.net/osf/manifest"
           xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
           targetNamespace="http://openscoreformat.sourceforge.net/osf/manifest"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
               schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
    <xs:element name="manifest">
         <xs:annotation>
              <xs:documentation>
Manifest for an OSF Package. The manifest records information for each file in the
package besides container.xml and manifest.xml, such as location, format type and
data associated with any (optional) signature for the package and its assets.
            </xs:documentation>
         </xs:annotation>
         <xs:complexType>
              <xs:sequence>
                   <xs:element name="assets">
                        <xs:complexType>
                             <xs:sequence>
                                  <xs:element name="asset" maxOccurs="unbounded">
                                       <xs:complexType>
                                            <xs:attribute name="name" type="xs:anyURI" use="requir
                                                 <xs:annotation>
                                                      <xs:documentation>
                                                 IRI (as defined in RFC3987) of the file relative
                                                      </xs:documentation>
                                                 </xs:annotation>
                                            </xs:attribute>
                                            <xs:attribute name="media-type" type="xs:anyURI" use="
                                                 <xs:annotation>
                                                      <xs:documentation>
                                                           Internet Media Type (MIME-Type)  as
                                                      </xs:documentation>
                                                 </xs:annotation>
                                            </xs:attribute>
                                            <xs:attribute name="size" type="xs:integer" use="optic
```

```
                <xs:annotation>
                    <xs:documentation>
                    Size of file (optional). Used when files a
                    And padded may have needed to have been ap
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>
            <xs:attribute name="Id" type="xs:ID"/>
            <xs:attribute name="signed" type="xs:boolean" use="opt
                <xs:annotation>
                    <xs:documentation>
                    The asset should be included in the digita
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>
            <xs:attribute name="c14n" type="xs:boolean" use="optio
                <xs:annotation>
                    <xs:documentation>
                    The asset should be canonicalized before s
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>
        </xs:complexType>
    </xs:element>
    <xs:element name="asset-group" minOccurs="0" maxOccurs="unbounded
        <xs:annotation>
            <xs:documentation>
                A group of assets in the package. Typically a gr
                encyption key, are processed together by the sec
                ever belong in one asset group
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
                <xs:element name="asset-reference">
```

```
                                                      <xs:complexType>
                                                            <xs:attribute name="ref" type="xs:an
                                                      </xs:complexType>
                                                </xs:element>
                                          </xs:sequence>
                                          <xs:attribute name="name" type="xs:anyURI" use="requir
                                    </xs:complexType>
                              </xs:element>
                        </xs:sequence>
                        <xs:attribute name="name" type="xs:anyURI" use="required"/>
                  </xs:complexType>
            </xs:element>
            <xs:element ref="ds:KeyInfo" minOccurs="0">
                  <xs:annotation>
                        <xs:documentation>
                              Public key used to sign package. Beneath here should be a X509Data
                              an BASE64 encoded X.509 certicate bearing the key. Whilst the sche
                              limitation, the OSF validation tool probably will.
                        </xs:documentation>
                  </xs:annotation>
            </xs:element>
            <xs:element ref="ds:Signature" minOccurs="0">
                  <xs:annotation>
                        <xs:documentation>
                              XML-SIG signature of and asset digests
                        </xs:documentation>
                  </xs:annotation>
            </xs:element>
      </xs:sequence>
</xs:complexType>
<!-- cross reference and key uniqueness tests -->
<xs:key name="asset">
      <xs:annotation>
            <xs:documentation>
                  Enforce unique asset names.
```

```
                        </xs:documentation>
                  </xs:annotation>
                  <xs:selector xpath="osfm:assets/osfm:asset"/>
                  <xs:field xpath="@name"/>
            </xs:key>
            <xs:key name="asset-group-name">
                  <xs:annotation>
                        <xs:documentation>
                              Enforce unique asset-group names.
                        </xs:documentation>
                  </xs:annotation>
                  <xs:selector xpath="osfm:assets/osfm:asset-group"/>
                  <xs:field xpath="@name"/>
            </xs:key>
            <xs:keyref name="asset-group" refer="osfm:asset">
                  <xs:annotation>
                        <xs:documentation>
                              Ensure that asset-ref corresponds to asset
                        </xs:documentation>
                  </xs:annotation>
                  <xs:selector xpath="osfm:assets/osfm:asset-group/osfm:asset-reference"/>
                  <xs:field xpath="@ref"/>
            </xs:keyref>
            <xs:unique name="asseet-reference">
                  <xs:annotation>
                        <xs:documentation>
                              Assets can only be referenced in one asset group
                              </xs:documentation>
                  </xs:annotation>
                  <xs:selector xpath="osfm:assets/osfm:asset-group/osfm:asset-reference"/>
                  <xs:field xpath="@ref"/>
            </xs:unique>
            <!-- cross references  checks  between things in the manifest that can be signed and references
            <xs:key name="signable-object">
                  <xs:annotation>
```

```
                        <xs:documentation>
                              Any object to which a digital signature can be applied (see xs:signature elem
                              This contains assets and the assets element
                        </xs:documentation>
                  </xs:annotation>
                  <xs:selector xpath="osfm:assets/osfm:asset|osfm:assets"/>
                  <xs:field xpath="@name"/>
            </xs:key>
      </xs:element>
</xs:schema>
```

7.3 META-INF/metadata.xml W3C XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
* Version 1.0  3/7/09
* Copyright © 2009 Yamaha Corporation
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above copyright
*       notice, this list of conditions and the following disclaimer in the
*       documentation and/or other materials provided with the distribution.
*     * Neither the name of the Yamaha Corporation nor the
*       names of its contributors may be used to endorse or promote products
*       derived from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY YAMAHA CORPORATION ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL YAMAHA CORPORATION BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:dcterms="http://purl.org/dc/terms/"
           xmlns:osfmeta="http://openscoreformat.sourceforge.net/osf/metadata"
           xmlns:dc="http://purl.org/dc/elements/1.1/"
           targetNamespace="http://openscoreformat.sourceforge.net/osf/metadata"
           elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:import namespace="http://purl.org/dc/terms/"
            schemaLocation="http://dublincore.org/schemas/xmls/qdc/dcterms.xsd"/>
    <xs:import namespace="http://purl.org/dc/elements/1.1/"
            schemaLocation="http://dublincore.org/schemas/xmls/qdc/dc.xsd"/>
    <xs:element name="app-ext-metadata" abstract="true"/>
    <xs:annotation>
          <xs:documentation>Abstract placeholder element to application-specific metadata extensions</xs:
    </xs:annotation>
    <xs:complexType name="meter-type">
          <xs:annotation>
                <xs:documentation>
                Meter (eg. time signature). Expressed as beats (eg. numerator, or number of beats per mea
                type (denominator)
                </xs:documentation>
          </xs:annotation>
          <xs:sequence>
                <xs:element name="beats" type="xs:string">
                      <xs:annotation>
                            <xs:documentation>Number of beats in each measure</xs:documentation>
                      </xs:annotation>
                </xs:element>
                <xs:element name="beat-type" type="xs:string">
                      <xs:annotation>
                            <xs:documentation>Type of beat 4 = quarter, 8 = 8th -note etc.</xs:documentat
```

```
                  </xs:annotation>
            </xs:element>
      </xs:sequence>
</xs:complexType>
<xs:simpleType name="opus-type">
      <xs:annotation>
            <xs:documentation>Opus number of a work</xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:integer"/>
</xs:simpleType>
<xs:complexType name="key-type">
      <xs:annotation>
            <xs:documentation>Key signature of work</xs:documentation>
      </xs:annotation>
      <xs:sequence>
            <xs:element name="root">
                  <xs:annotation>
                        <xs:documentation>
        Root key of key signature. If work is atonal or has no obvious key-centre, select 'none'
                        </xs:documentation>
                  </xs:annotation>
                  <xs:simpleType>
                        <xs:restriction base="xs:string">
                              <xs:enumeration value="C"/>
                              <xs:enumeration value="D"/>
                              <xs:enumeration value="E"/>
                              <xs:enumeration value="F"/>
                              <xs:enumeration value="G"/>
                              <xs:enumeration value="A"/>
                              <xs:enumeration value="B"/>
                              <xs:enumeration value="none"/>
                        </xs:restriction>
                  </xs:simpleType>
            </xs:element>
            <xs:element name="accidental" minOccurs="0">
```

```
                        <xs:annotation>
                              <xs:documentation>
             Accidentals applied to root key of key signature. Omit if key is natura
                              </xs:documentation>
                        </xs:annotation>
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:enumeration value="sharp"/>
                                    <xs:enumeration value="flat"/>
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
                  <xs:element name="mode">
                        <xs:annotation>
                              <xs:documentation>Diatonic mode of key signature</xs:documentation>
                        </xs:annotation>
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:enumeration value="minor"/>
                                    <xs:enumeration value="major"/>
                                    <xs:enumeration value="dorian"/>
                                    <xs:enumeration value="phrygian"/>
                                    <xs:enumeration value="lydian"/>
                                    <xs:enumeration value="mixolydian"/>
                                    <xs:enumeration value="aeolian"/>
                                    <xs:enumeration value="ionian"/>
                                    <xs:enumeration value="locrian"/>
                                    <xs:enumeration value="none"/>
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
      <xs:element name="osf-package-metadata-baseline">
            <xs:annotation>
```

```
                  <xs:documentation>Container of metadata terms</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="dc:title" maxOccurs="unbounded"/>
                        <xs:element ref="dc:creator" maxOccurs="unbounded"/>
                        <xs:element ref="dc:description" maxOccurs="unbounded"/>
                        <xs:element ref="dc:publisher" maxOccurs="unbounded"/>
                        <xs:element ref="dc:date" maxOccurs="unbounded"/>
                        <xs:element ref="dc:type"/>
                        <xs:element ref="dc:format" maxOccurs="unbounded"/>
                        <xs:element ref="dc:identifier"/>
                        <xs:element ref="dc:source"/>
                        <xs:element ref="dc:language" maxOccurs="unbounded"/>
                        <xs:element ref="dc:relation" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="dcterms:audience" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="dcterms:coverage" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element ref="dc:rights" maxOccurs="unbounded"/>
                        <xs:element ref="dcterms:mediator" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="meter" type="osfmeta:meter-type" maxOccurs="unbounded"/>
                        <xs:element name="opus" type="osfmeta:opus-type" minOccurs="0" maxOccurs="unbounded
                        <xs:element name="key" type="osfmeta:key-type" maxOccurs="unbounded"/>
                        <xs:element ref="osfmeta:app-ext-metadata" minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

7.4 Example `META-INF/manifest.xml` with a digital signature

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
* Copyright (c) 2008-2009 Yamaha Corporation
* All rights reserved.
```

```
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above copyright
*       notice, this list of conditions and the following disclaimer in the
*       documentation and/or other materials provided with the distribution.
*     * Neither the name of the <organization> nor the
*       names of its contributors may be used to endorse or promote products
*       derived from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY YAMAHA CORPORATION ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL YAMAHA CORPORATION BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
-->
<!-- OSF Example Package - Manifest.xml.  Note that all package files are listed including the manifest i
     This package contains two renditions - the default and an alternate (application specific) one.
-->
<osfm:manifest xmlns:osfm="http://openscoreformat.sourceforge.net/osf/manifest"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xsi:schemaLocation="http://openscoreformat.sourceforge.net/osf/manifest
                          http://openscoreformat.sourceforge.net/1.0/manifest.xsd">
    <!-- Package metadata -->
    <osfm:assets name="ExampleManifest">
          <osfm:asset name="META-INF/Container.xml" media-type="text/xml" signed="true" c14n="true"/>
          <osfm:asset name="META-INF/Manifest.xml" media-type="text/xml" signed="true" c14n="true"/>
          <osfm:asset name="META-INF/Metadata.xml" media-type="text/xml" signed="true" c14n="true"/>
          <!-- default rendition -->
          <osfm:asset name="Default/Score.xml" media-type="application/application-osf-score-pvg-profile"
          signed="true" c14n="true"/>
          <!-- altnernate rendition -->
          <osfm:asset name="Alternate/Enhancement.xml" media-type="applicatication/myAppScoreEnhancement"
          signed="true" c14n="true"/>
          <osfm:asset name="Image1.png" media-type="image/png" signed="true" c14n="false"/>
          <osfm:asset name="Image2.png" media-type="image/png" signed="true" c14n="false"/>
          <osfm:asset name="Image3.png" media-type="image/png" signed="true" c14n="false"/>
          <osfm:asset name="Image4.png" media-type="image/png" signed="true" c14n="false"/>
          <osfm:asset-group name="AlternateRendition">
                <osfm:asset-reference ref="Alternate/Enhancement.xml"/>
                <osfm:asset-reference ref="Image1.png"/>
                <osfm:asset-reference ref="Image2.png"/>
                <osfm:asset-reference ref="Image3.png"/>
                <osfm:asset-reference ref="Image4.png"/>
          </osfm:asset-group>
    </osfm:assets>
    <ds:Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
          <ds:SignedInfo>
                <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                <ds:Reference URI="../META-INF/Container.xml">
                      <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-ds:Signa
                      </ds:Transforms>
                      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                      <ds:DigestValue>2a3Xy5hSLQMumef8YbNwjo/eIEY=</ds:DigestValue>
                </ds:Reference>
                <ds:Reference URI="../META-INF/Manifest.xml">
                      <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-ds:Signa
                      </ds:Transforms>
                      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <ds:DigestValue>wWhTFSOTE/c4Q2vUGG4QgU3oX0k=</ds:DigestValue>
            </ds:Reference>
            <ds:Reference URI="../META-INF/Metadata.xml">
                <ds:Transforms>
                    <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-ds:Signa
                </ds:Transforms>
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <ds:DigestValue>wiHrbeC6b6aOCc1E5S4y+DO/hUg=</ds:DigestValue>
            </ds:Reference>
            <ds:Reference URI="../Default/Score.xml">
                <ds:Transforms>
                    <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-ds:Signa
                </ds:Transforms>
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <ds:DigestValue>Vby11ZAi/KwtA57x1dhwVgk+DL8=</ds:DigestValue>
            </ds:Reference>
            <ds:Reference URI="../Alternate/Enhancement.xml">
                <ds:Transforms>
                    <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-ds:Signa
                </ds:Transforms>
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <ds:DigestValue>Vby11ZAi/KwtA57x1dhwVgk+DL8=</ds:DigestValue>
            </ds:Reference>
            <ds:Reference URI="../Image1.png">
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <ds:DigestValue>wDlUs4ZbwrzwTzdd3mxj5fyLKC8=</ds:DigestValue>
            </ds:Reference>
            <ds:Reference URI="../Image2.png">
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <ds:DigestValue>Pk9ZVSbpn8olwaM8UElhEdko5Mk=</ds:DigestValue>
            </ds:Reference>
            <ds:Reference URI="../Image3.png">
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <ds:DigestValue>4c/B4wfPkwCIIAT2CJeUO+d4UmY=</ds:DigestValue>
            </ds:Reference>
```

```
                <ds:Reference URI="../Image4.png">
                      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                      <ds:DigestValue>CKo7sW5fXRNJWDVHdb5oK/CRt90=</ds:DigestValue>
                </ds:Reference>
                <ds:Reference URI="">
                      <ds:Transforms>
                            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-ds:Signa
                      </ds:Transforms>
                      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                      <ds:DigestValue>6jHiD24vvcpBsVyrrQMQgaSs1Zs=</ds:DigestValue>
                </ds:Reference>
          </ds:SignedInfo>
          <ds:SignatureValue>
                KgqGUt7U97Kg4QinaB7KgiL7kBzFnsgLfRbHpQNhrvFpwkUuIq/gQ75klgLNV4n/SX9PV5fi7l5RW/sA/p32FAyaP
                TWn53jMwt1maPFKKsWAN4g9W4NPkABZaUXHZ6KQnAyFo6wEJ6wXaAHKxLbFyxPSKycoB/lLRk4=
          </ds:SignatureValue>
          <ds:KeyInfo>
                <ds:X509Data>
                      <ds:X509Certificate>
          MIICpDCCAg2gAwIBAgIJALl19pxX6z+bMA0GCSqGSIb3DQEBBQUAMEExCzAJBgNVBAYTAlVLMRAwDgYDVQQIEwdFbmdsYW
          5kMQ8wDQYDVQQHEwZMb25kb24xDzANBgNVBAoTBllhbWFoYTAeFw0wODExMTAxMDU2MTRaFw0xMTExMTAxMDU2MTRaMEEx
          CzAJBgNVBAYTAlVLMRAwDgYDVQQIEwdFbmdsYW5kMQ8wDQYDVQQHEwZMb25kb24xDzANBgNVBAoTBllhbWFoYTCBnzANBg
          kqhkiG9w0BAQEFAAOBjQAwgYkCgYEAv+9tcj7Msu0H3b9qLiYmQb4gG2pzBj5BmNxZYO1jvC43OmxcZsHwSAXMqXvVeBuC
          0rt9i2I9zE7fu/oDtx/SdPxh8aZqS5QR90Ue9jkTj/zboq85a8eUf0GuEKFV2D1fTkmlfjuzcKq1t1pPNpyEZwVvozSfE5
          Va3jMHtAy2ubcCAwEAAaOBozCBoDAdBgNVHQ4EFgQU4mk2V/t9rdLcT4add1ndctlI9tkwcQYDVR0jBGowaIAU4mk2V/t9
          rdLcT4add1ndctlI9tmhRaRDMEExCzAJBgNVBAYTAlVLMRAwDgYDVQQIEwdFbmdsYW5kMQ8wDQYDVQQHEwZMb25kb24xDz
          ANBgNVBAoTBllhbWFoYYIJALl19pxX6z+bMAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQEFBQADgYEADbQJ5BUT7BHpa4ND
          j3Uv9n+rF6FD5rSZnTxIvVs9V2tyqweA3UXgRPGKEYhHO7ZUoBBR4LJxRBRdYh3zZlrQkxduSaF3ghmY4V6o9080h3kFjs
          a5N2k1vcCljezck0n0Ru4JH0Lbtj0XXjKJMtUnT9a/hRVPxAtOzLRzNlPsFJ8=
                      </ds:X509Certificate>
                </ds:X509Data>
          </ds:KeyInfo>
    </ds:Signature>
</osfm:manifest>
```

7.5 Open Score Format Base-line Metadata Profile Dublin Core Element requirement levels

The table below specifies the Dublin Core metadata terms that should be present in `META-INF/metadata.xml`.

Terms in the Element Refinement column are defined in the *Qualified Dublin Core*, and in many cases enhance th
Dublin Core (Dublin Core Element column). Both Simple Dublin Core Elements and Qualified Dublin Core Elemen
`INFO/metadata.xml`. Refer to *The Dublin Core Metadata Initiative: Using the Dublin Core – The Elements* [20] fo

The *OSF Baseline Metadata Profile* requires certain metadata terms to be present as described in the table below.
RECOMMENDED because their meaning is ambiguous or because an element refinement from the *Qualified Dub*

Dublin Core Element	Element Refinement[13]	Element Encoding Scheme	Requirement Le
`dc:title`	-	Any	**Required**
	`dcterms:alternative`		**Optional**
`dc:creator`	-	Any	**Required**
`dc:description`	-	Any	**Required**
	`dcterms:tableOfContents`		**Recommended**
	`dcterms:abstract`		**Optional**
`dc:publisher`	-	Any	**Required**
`dc:contributor`	-	Any	**Optional**
`dc:date`	--	W3C DTF	**Not recommend**
	`dcterms:created`	DCMI Period	**Required**
	`dcterms:valid`		**Optional**
	`dcterms:available`		**Optional**
	`dcterms:issued`		**Required**

[13] Element refinements are elements that convey extended semantics that can be used in place of a Dublin Core element.

	dcterms:modified		Required
	dcterms:dateCopyrighted		Required
dc:type	-	DCMI Type vocabulary[14]	Required
dc:format	-	Any	Not recommenc
	dcterms:extent	Any	Optional
	dcterms:medium	IMT [4]	Required
dc:identifier	-	URI	Required
dc:source	-	URI	Required
dc:language	-	RFC4646 [14]	Required
dc:relation	-	URI	Not recommenc
	dcterms:isVersionOf		Recommended
	dcterms:hasVersion		Recommended
	dcterms:isReplacedBy		Recommended
	dcterms:replaces		Recommended
	dcterms:isRequiredBy		Optional
	dcterms::requires		Optional
	dcterms:isPartOf		Recommended
	dcterms:hasPart		Recommended
	dcterms:isReferencedBy		Recommended
	dcterms:references		Recommended
	dcterms:isFormatOf		Optional
	dcterms:hasFormat		Optional

[14] The most appropriate type from the DCMI vocabulary is InteractiveResource - http://purl.org/dc/dcmitype/InteractiveResource

	dcterms:isBasisOf		**Optional**
	dcterms:isBasedOn		**Optional**
dc:coverage	-	Any	**Optional**
	dcterms:spatial	Any	**Optional**
	dcterms:temporal	W3C-DTF DCMI Period	**Recommended**
dc:audience	-	Any	**Optional**
	dcterms:mediator	URI	**Optional**
	dcterms:educationLevel	Any	**Optional**
dc:rights	-	Any	**Optional**

Table 6: Use of Qualified Dublin Core elements