Public Record Office Victoria

SPECIFICATION

PROS 19/05 S4: CONSTRUCTING VEOs

Version number: 1.0
Issue Date: 15 April 2020
Expiry Date: 29 August 2029

About this Specification

Digital records which have permanent value must be transferred to PROV as VERS Encapsulated Objects (VEOs), at a time agreed between PROV and the public office.

This Specification is for technical developers building systems and tools to construct VEOs. It describes how to construct VEOs which meet Public Record Office Victoria (PROV) requirements. *Note – the actual VERS code can be downloaded from the PROV website.*

The diagram below shows the relationship between this Specification and related documents.

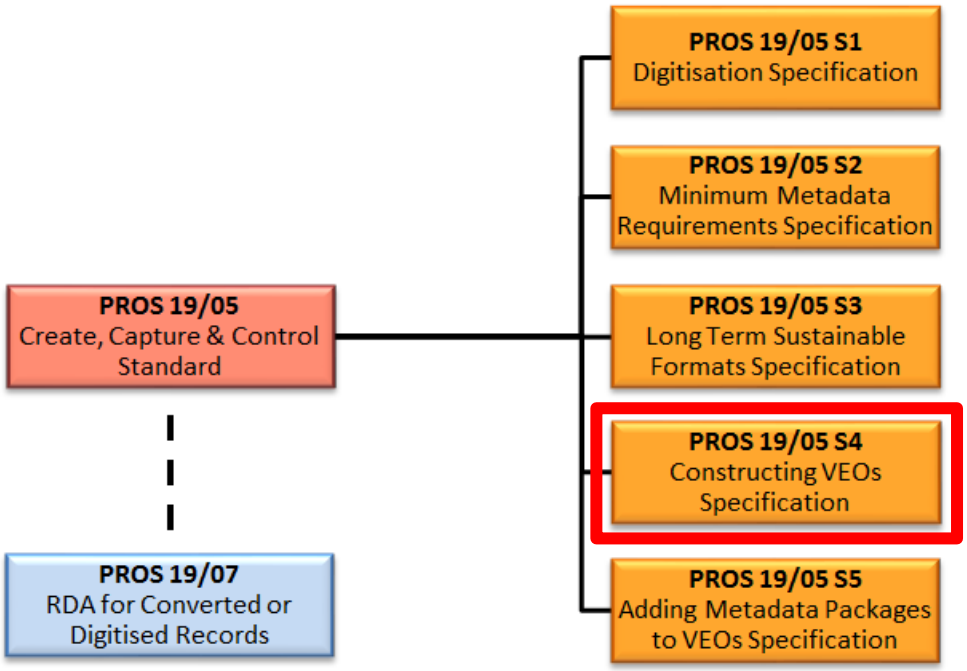


Table of Contents

1 Introduction

1.1 Authority of Standards and Specifications

Under section 12 of the *Public Records Act 1973,* the Keeper of Public Records ('the Keeper') is responsible for the establishment of Standards for the efficient management of publicHeads of public offices are responsible under section 13b of the *Public Records Act 1973* for carrying out a program of efficient management of public records. The program of records management needs to cover all records created by the public office, in all formats, media and systems across the organisation.

The Standards and Specifications support the Victorian Electronic Record Strategy (VERS) Digital Forever 2018-2021[1], which is designed to ensure the creation, capture and preservation of authentic, complete and meaningful digital records.

This Specification is part of the *PROS19/05 Create, Capture and Control Standard*. This Specification, as varied or amended from time to time, shall have effect for a period of ten (10) years from the date of issue unless revoked prior to that date.

1.2 Obligation

It is mandatory for all Victorian public offices to follow the principles and comply with the requirements of the Standards issued by the Keeper. Some of the Standards have Specifications, which provide detailed technical requirements that must be complied with by Victorian public offices.

1.3 Applying this Specification

All Victorian public offices must transfer permanent value digital records to PROV as VEOs.

This Specification is for technical developers building systems and tools to construct VEOs. It outlines a process for constructing VEOs which will meet PROV requirements.

This Specification supersedes *PROS 15/03 S1 Constructing VERS Encapsulated Objects (VEOs*), which is now revoked. This new Specification does not change the technical requirements for constructing VEOs.

[1] The previous *PROS15/03 Standard for the encapsulation of digital records* has been revoked and the requirements have now been included in the *PROS19/05 Create, Capture and Control Standard* and associated Specifications.

2 Constructing VEOs

2.1 Overview

Conformant VEO construction is presented as a process, to make the requirements as clear as possible. Any alternative process may be used to construct VEOs, provided the resulting ZIP files are identical to that produced by the process presented in this Specification.

The process can be summarised as:

- create a directory to contain the VEO information
- create subdirectories within the VEO directory to contain the record content
- copy the record content into the subdirectories
- create an XML file describing the logical organisation of the record content and the metadata describing the record.

When creating a VEO, creators have the following freedoms:

- there are no restrictions on the naming or organisation of the content within the subdirectories. Any number of subdirectories may be present
- the content can be logically organised into an arbitrary hierarchy (or a completely flat list). There is no requirement that the logical hierarchy reflect the physical organisation of the content in the subdirectories
- content can be described using arbitrary metadata packages. The packages must be expressed as XML and preferably, in RDF.

2.2 General Standard References

This specification references the following external documents:

2.2.1 Extensible Markup Language (XML)

XML is used to represent the metadata, and is defined in Extensible Markup Language (XML) 1.0 (Fifth Edition).

2.2.2 ISO 8601 – Date and Time Formats

Dates and times are to be represented using the W3C profile of ISO 8601.

Unless otherwise noted, date/time values may be specified to the:

- year
- year and month
- year, month and date
- year, month, date and time

Unwarranted precision must not be used (e.g. if a date is only known to a month, only the year and month must be specified). **Fractional seconds must not be used.**

2.2.3 Base64

The Base64 encoding is used to represent binary data as text. In this Specification, Base64 is used to represent hash values, digital signatures, and X.509 certificates.

The Base64 encoding is defined in RFC2045.

2.2.4 Certificates

X.509 certificates are used to represent the information necessary to validate the digital signatures.

X.509 certificates are defined in ITU-T Recommendation X.509 (2005): Information Technology – Open Systems Interconnection – The Directory: Authentication Framework.

An alternative source of information can be found in RFC5280.

2.2.5 ASN.1 (Abstract Syntax Notation One)

ASN.1 is used to represent the X.509 certificates.

ASN.1 is defined in ITU-T Recommendation X.680 (11/2008): Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.

This document is also published as ISO/IEC 8824-1.

2.2.6 DER (Distinguished Encoding Rules)

The X.509 certificates, represented in ASN.1, are expressed using the Distinguished Encoding Rules (DER).

The Distinguished Encoding Rules are defined in ITU-T Recommendation X.690 (2002): Information technology- ASN.1 encoding rules (http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf).

This document is also published as ISO/IEC 8825-1.

2.3 VEO overview

A VEO consists of a ZIP file that contains specific files and subdirectories:

- Content subdirectories. These contain the information encapsulated in the VEO.
- VEOContent.xml. This file logically organises the information in the content subdirectories and associates metadata with the component parts.
- VEOHistory.xml. This file contains an event log that documents the history of this VEO.
- VEOContentSignature?.xml and VEOHistorySignature?.xml files. These files contain digital signatures that allow detection of corruption of the VEOContent.xml and VEOHistory.xml files.
- VEOReadMe.txt. This file contains a text description of a VEO and its constituent parts.

The following sections describe a process for creating a VEO.

Step 1: Create a VEO directory

A directory must be created to contain the content and metadata of the VEO.
The directory must be named with the extension '.veo' (e.g. xyzzy.veo).

Step 2: Copy the content into the VEO directory

Copy the information (content files) to be contained in the VEO into subdirectories of the VEO directory.

Creators can:

- use any number of subdirectories to contain the information
- arrange the information in any manner within the subdirectories
- name the subdirectories using any legal name.

We suggest that the information is copied in the structure that it originally occurs, and that additional subdirectories are created for any added information (e.g. migrated content or documentation about the information).

Step 3: Copy the VEOReadme.txt file into the VEO directory

Copy the VEOReadme.txt file into the VEO directory. The contents of this file are given below and should not be changed.

The purpose of this file is to provide a human readable description of a VEO and the standard arrangement of the information within it.

Descriptive information (e.g. documentation) about the information contents of the VEO must not be added to the VEOReadme.txt file. Instead, it should be added to subdirectories of the VEO directory.
For example, if the VEO contained a SIARD capture of a relational database, the SIARD files might be contained in a subdirectory 'SIARD'. Any documentation about the database might be captured in a subdirectory 'Documentation' within the SIARD directory.

Text of VEOReadme.txt file

The Standard VEOReadme.txt file can be downloaded from the PROV website

<Start of VEOReadme.txt File>

```
This zip file is a VERS Encapsulated Object (VEO). VEO files are
specified in Public Record Office Victoria (PROV) Standard
PROS 15/03 "Long term management of Electronic Rcords". As at 2015 this
Standard was available from http://prov.vic.gov.au/government/vers

This VEOReadme.txt file contains a summary of the information in
PROS 15/03.

This zip file contains a collection of information that it is desired to
keep accessible for a long period. The content of this information is
contained in subdirectories of this directory. The metadata that
organises and describes the information is contained in the
VEOContent.xml file.

The VEOContent.xml file contains one or more Information Objects – each
of which is a logical collection of information. Information Objects
may contain one or more Information Pieces. An Information Piece
represents a piece of information contained in this VEO. An Information
Piece includes references to at least one physical file (a Content File)
that actually represent the content. If more than one Content File is
present, this represents the same information expressed using different
software formats.

The XML elements in the VEOContent file are:
```

```
   *  VEOVersion. The version of this Standard. This should be '3.0'.
   *  HashFunctionAlgorithm. The hash function used to calculate the
      hash values stored in HashValue (see below)
   *  InformationObject. A logical collection of information in this
      VEO.
   *  InformationObjectType. A text label describing the purpose of
      this Information Object.
   *  InformationObjectDepth. If the VEO contains more than one
      Information Object, the Information Objects may be organised in
      a tree. The sequence of Information Objects in the
      VEOContent.xml file will form a depth first traversal of this
      tree, and InformationObjectDepth is the depth of this
      particular Information Object in the tree.
   *  MetadataPackage. This is a collection of metadata describing
      the Information Object. An Information Object may have multiple
      Metadata Packages.
   *  MetadataSchemaIdentifier. This is used to identify the type of
      this Metadata Package.
   *  MetadataSyntaxIdentifier. This is used to identify the way of
      representing the metadata package in XML. We encourage the use
      of RDF.
   *  InformationPiece. This is a piece of information content within
      the Information Object.
   *  Label. This is a text string that labels the Information Piece.
   *  ContentFile. This represents a specific file in the VEO that
      represents the content of the InformationPiece.
   *  ContentFilePathName. This is the file name of the file that
      contains the content. It is relative to this directory.
   *  HashValue. The hash value resulting from applying the specified
      hash function (see HashFunctionAlgorithm above) to the sequence
      of octets forming the file.

   The VEOContentSignature?.xml (where '?' is a number) files each contain
   a digital signature that allows detection of corruption of the
   VEOContent.xml file (and because the VEOContent.xml file includes hash
   values of the content files, corruption of the content files as well).
   The signature is generated by reading the VEOContent.xml file as a
   sequence of octets and applying specified signature algorithm. The
   contents of a VEOContentSignaturen.xml file are:
   *  SignatureAlgorithm. This identifies the hash algorithm and the
      digital signature algorithm used to generate the signature.
   *  SignatureDateTime. The date and time the signature was applied.
      Expressed in ISO8601.
   *  Signer. A text string naming the person or organisation that
      created the digital signature
   *  Signature. The resulting signature, encoded as Base64.
   *  CertificateValue. An X.509 DER encoded certificate encoded as
   Base64. The first certificate contains the public key used to validate
   the signature. The second certificate contains the public key used to
   validate the first certificate (and so on). The last certificate must
   be self signed.

   The VEOHistory.xml file contains a summary of events in the life of
   this VEO. The elements in this file are:
   *  VEOVersion. The version of this Standard. This should be '3.0'.
   *  Event. A collection of information about an event in the life
      of this VEO.
   *  EventDateTime. The date and time the event occurred. Expressed
      in ISO8601.
   *  EventType. The a text string labelling the type of the event.
   *  Initiator. The person or organisation that authorised or
      initiated this event.
   *  Description. A text description of the event.
   *  Errors. Any error resulting from this event.

   The VEOHistorySignaturen.xml (where 'n' is a number) files each contain
   a digital signature that allows detection of corruption of the
   VEOHistory.xml file. The contents and method of generation of a
   VEOHistorySignature file is identical to a VEOContentSignature file.
```

<End of VEOReadme.txt File>

Step 4: Create the VEOContent.xml file in the VEO directory

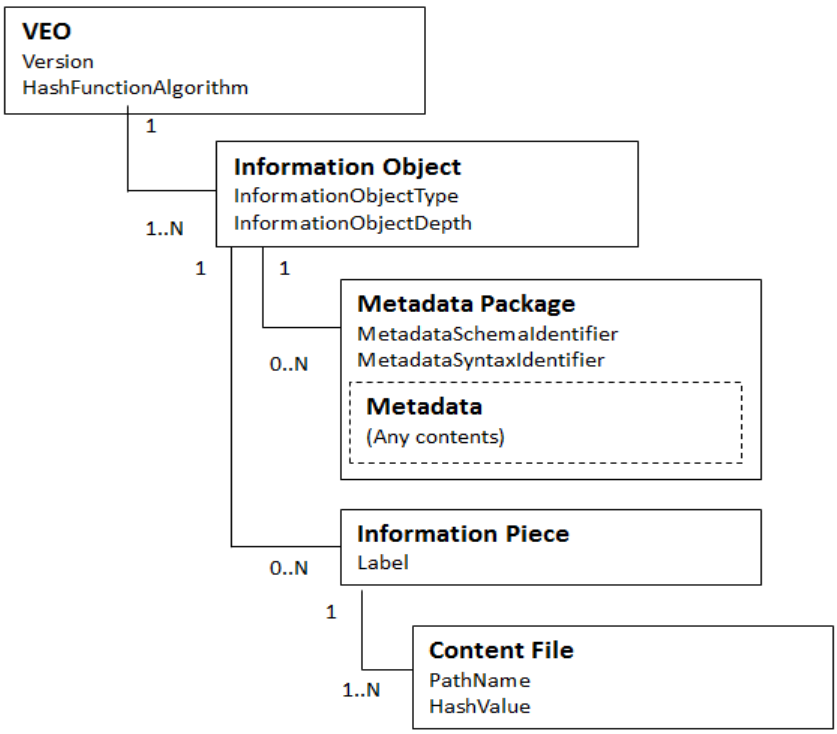
Each VEO must contain a VEOContent.xml file in the top level of the VEO directory.

The VEOContent.xml file is essentially a manifest of the information contained in the subdirectories. It performs the following functions:

- lists every content file contained in the content subdirectories of the VEO. Every file listed in the VEOContent.xml file must be present in the VEO, and every file contained in the content subdirectories must be listed in the VEOContent.xml file.
- contains a cryptographic hash value of every content file contained in the VEO. This allows corruption of these files to be detected. The hash function used to generate these values is identified in the VEOContent.xml file2.
- associates alternative representations of the same information together (e.g. Word and PDF files that represent the same content).
- logically organises the information in the content subdirectories and, optionally, associates metadata with the logical organisation. This logical organisation need not have any relation to the physical arrangement of the content files in the subdirectories.

The VEOContent.xml file contains one or more Information Objects. If more than one Information Object is present, they may be arranged in a hierarchy. Each Information Object contains zero or more Metadata Packages (the first Information Object must contain at least one Metadata Package) and contains zero or more Information Pieces. Each Information Piece contains one or more Content Files – each Content File represents alternative representations of the same content.

The following diagram shows the arrangement of the information within a VEOContent.xml file, and the table gives more information about the XML elements.



[2] Changes to these hash values are prevented by digitally signing the VEOContent.xml file, as described in a later section.

Table 1 VEOContent.xml Structure Elements

Element Name	*Occurrences*		*Purpose*	*Value*
	Min	*Max*		
Version	1	1	Identifies the version number this VEO is conformant to	3.0
HashFunction-Algorithm	1	1	Identifies the hash function used to generate the hash values contained in the ContentFile entities	Valid values are 'SHA-1', 'SHA-256', 'SHA-384', 'SHA-512'[3]
InformationObject-Type	1	1	Identifies the type of this Information Object	User defined string
InformationObject-Depth	1	1	If there is more than one Information Object in the VEO, and these Information Objects are organised hierarchically, this is the depth of this Information Object in the hierarchical tree.	If the VEO only contains one Information Object, or the Information Objects are not arranged hierarchically, this element has the integer value 0. If the VEO contains more than one Information Object, and the Information Objects are arranged hierarchically, this element has an integer value indicating the depth of the Information Object in the tree. The root of the tree has a depth of '1'.
MetadataSchema-Identifier	1	1	Identifies the schema of the metadata package	The URI of the schema or written definition of the metadata. It is not necessary that this URI resolves to an actual web object.
MetadataSyntax-Identifier	1	1	Identifies the syntax used to express the metadata package	The URI of the specification of the syntax. For metadata expressed as RDF, this element has the value 'http://www.w3.org/1999/02/22-rdf-syntax-ns'. It is not necessary that this URI resolves to an actual web object.
Label	0	1	A text string that distinguish this Information Piece from all other Information Pieces in the Information Object	User defined string
PathName	1	1	The relative file name of a content file in the VEO subdirectory	Path name relative to the root directory of the VEO. Filename separators are forward slashes ('/')
HashValue	1	1	The result of applying the hash function (identified by Hash Function Algorithm) to the file identified by ContentPathName.	Base64 encoded binary array

[3] PROV does not allow the use of the MD-5 hash function in this version of the Standard as this is considered to be insecure. SHA-1 is becoming insecure, and SHA-1 should only be used if you do not have access to one of the versions of SHA-2 (i.e. 'SHA-256', 'SHA-384', or 'SHA-512').

The XML schema for the VEOContent.xml file is:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:vers="http://www.prov.vic.gov.au/VERS"
      targetNamespace="http://www.prov.vic.gov.au/VERS"
      elementFormDefault="qualified">

<xs:element name="VEOContent">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Version" type="xs:string" default="3.0"/>
      <xs:element name="HashFunctionAlgorithm" type="xs:string" />
      <xs:element ref="vers:InformationObject" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="InformationObject">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="InformationObjectType" type="xs:string"/>
      <xs:element name="InformationObjectDepth" type="xs:nonNegativeInteger"/>
      <xs:element ref="vers:MetadataPackage" minOccurs="0"
             maxOccurs="unbounded"/>
      <xs:element ref="vers:InformationPiece" minOccurs="0"
             maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="MetadataPackage">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="MetadataSchemaIdentifier" type="xs:string"/>
      <xs:element name="MetadataSyntaxIdentifier" type="xs:string"/>
      <xs:any processContents="lax" minOccurs="1" maxOccurs="unbounded"/>
            <!-- any RDF -->
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="InformationPiece">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Label" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="vers:ContentFile" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="ContentFile">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="PathName" type="xs:string"/>
      <xs:element name="HashValue" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Number of Information Objects in a VEO

A VEO must contain at least one Information Object.
If a VEO only contains one Information Object, the InformationObjectDepth element of that Information Object must have the value 0.

Unstructured multiple Information Objects

A VEO may contain multiple Information Objects. If these Information Objects are not organised in a tree, the InformationObjectDepth element of each Information Object must have the value 0.

Arranging multiple Information Objects

If a VEO contains more than one Information Object, the Information Objects may be arranged into a tree. The first Information Object is the root of the tree and its InformationObjectDepth element must have the value 1. The remaining Information Objects are ordered by a depth first traversal of the tree and the value of each InformationObjectDepth element is the depth of that Information Object in the tree.

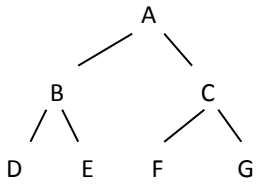
An algorithm to perform a depth first traversal of a tree is:

```
DepthFirstTraversal(node, depth)
   Output(node, depth)
   For each child of this node
         DepthFirstTraversal(child, depth+1)
```

This algorithm is called by:

```
DepthFirstTraversal(rootnode, 1)
```

Applied to this tree:



a depth first traversal will produce the following output (the figures in the brackets are the values of the InformationObjectDepth):

A(1), B(2), D(3), E(3), C(2), F(3), G(3)

Information Object Types

An information object type is simply a label chosen by the creator of the VEO to suggest indicate the purpose of the Information Object. This specification does not define a set of Information Object types[4].

[4] If only one Information Object is present in a VEO, the string 'Record' can be as the Information Object Type.

Metadata

The first Information Object in a VEO must at least contain one of the standard metadata packages defined in *PROS 19/05 Specification 5: Adding Metadata Packages to VEOs*.

Any Information Object (including the first Information Object in a VEO) may contain as many metadata packages as required. The metadata must be expressed as XML.

A URI identifying the metadata schema must be contained in the 'MetadataSchemaIdentifier' element, and a URI identifying the syntax of the metadata must be contained in the 'MetadataSyntaxIdentifier' element. More information about these URIs are contained in *PROS 19/05 Specification 5*.

References to content files

Each content file contained in a subdirectory of the VEO must be referred to in a ContentFile element in the VEOContent.xml file.

The PathName element contains the file name of the content file (relative to the VEO directory). For example, if a content file has the name 'DataFile1.doc' and it is in the 'Data' subdirectory of the VEO, the PathName element has the value 'Data/DataFile1.doc'.

Calculating the hash values of content files

Each content file referenced in a ContentFile must have an associated hash value in order to detect corruption of the content file.

The hash value is generated by passing each octet of the content file in sequence to the hash algorithm identified by the 'HashFunctionAlgorithm' element. The algorithm to calculate a hash value is:

```
Open content file as sequence of octets
Until end of file
   Read octet
   HashFunctionAddOctet(octet)
Get hash value from HashFunction()
Close content file
```

The resulting byte array is encoded as Base64 for inclusion in the VEO content file.

Step 5: Digitally signing the VEOContent.xml file

After the VEOContent.xml file is created it must be digitally signed at least once. The signatures will detect corruption of the VEOContent.xml file and the referenced content files[5]. Each signature and the certificates necessary to validate that signature must be placed in a VEOContentSignature?.xml file.

A VEOContentSignature?.xml file is an XML file. The first such file in the VEO is named 'VEOContentSignature1.xml'. If there is more than one signature file, subsequent files will be named 'VEOContentSignature2.xml', 'VEOContentSignature3.xml', and so on.

The following diagram shows the arrangement of the information within a VEOContentSignature?.xml file and the table gives more information about the XML elements.

[5] The referenced content files are protected because the VEOContent.xml file includes the hash value of the content file.

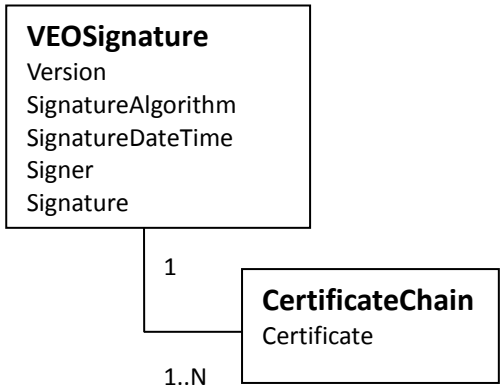


Table 2 VEOContentSignature and VEOHistorySignature Structure Elements

Element Name	***Occurrences***		***Purpose***	***Value***
	Min	***Max***		
Version	1	1	Identifies the version number this VEO is conformant to	3.0
SignatureAlgorithm	1	1	Identifies the algorithms used to generate the signature. – the hash algorithm and the signature algorithm	Valid values are 'SHA224withDSA', 'SHA224withRSA', 'SHA256withRSA', 'SHA256withDSA', 'SHA256withECDSA', 'SHA384withRSA', 'SHA384withECDSA', 'SHA512withRSA', 'SHA512withECDSA', SHA1withDSA', or 'SHA1withRSA'[6]
SignatureDateTime	1	1	States the date and time the signature was applied	A text string formatted according to ISO-8061
Signer	1	1	Identifies the person or organisation that applied the signature	A text string. A person is preferable to an organisation
Signature	1	1	The digital signature itself	A Base64 encoded binary value.
Certificate	1	1	A certificate used to validate the digital signature (or another certificate)	An X.509 certificate encoded using the Distinguished Encoding Rules (DER). The certificate is encoded using Base64.

[6] PROV does not allow the use of the MD-5 hash function in this version of the Standard as this is considered to be insecure. SHA-1 is becoming insecure, and SHA-1 should only be used if you do not have access to one of the versions of SHA-2 (i.e. 'SHA-256', 'SHA-384', or 'SHA-512').

The XML schema for the VEOContentSignature.xml and VEOHistorySignature files is:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:vers="http://www.prov.vic.gov.au/VERS"
      targetNamespace="http://www.prov.vic.gov.au/VERS"
      elementFormDefault="qualified">

<xs:element name="SignatureBlock">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Version" type="xs:string"/>
      <xs:element name="SignatureAlgorithm" type="xs:string"/>
      <xs:element name="SignatureDateTime" type="xs:dateTime"/>
      <xs:element name="Signer" type="xs:string"/>
      <xs:element name="Signature" type="xs:string"/>
      <xs:element ref="vers:CertificateChain" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="CertificateChain">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Certificate" type="xs:string" minOccurs="1"
          maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Calculating the signature

The signature is generated by passing each octet of the VEOContent.xml or VEOHistory.xml file in sequence to the signature algorithm identified by the 'SignatureAlgorithm' element. A typical algorithm to calculate the digital signature is:

```
Open VEOContent.xml file as sequence of octets
Until end of file
   Read octet
   AddOctet(octet)
CalculateSignature(PrivateKey)
Close VeoContent.xml file
```

The byte array generated by the signature algorithm is encoded as Base64 in the XML file.

Order of the certificates in the Certificate Chain

A Certificate Chain contains a sequence of certificates. The first certificate in the sequence contains the public key matching the private key used to calculate the signature. The second certificate contains the public key used to validate the first certificate (and so on). The last certificate must be self signed.

Step 6: Create the VEOHistory.xml file

The VEOHistory.xml file contains a history of the VEO since creation. A VEOHistory.xml file contains a sequence of events. Each event contains:

- The date and time the event occurred
- The person and organisation responsible for carrying out the event
- a description of the event
- any errors resulting from the event.

Each VEO will have a VEOHistory.xml file with at least one event (the creation of the VEO).

The following diagram shows the arrangement of the information within a VEOHistory.xml file, and the table gives more information about the XML elements.

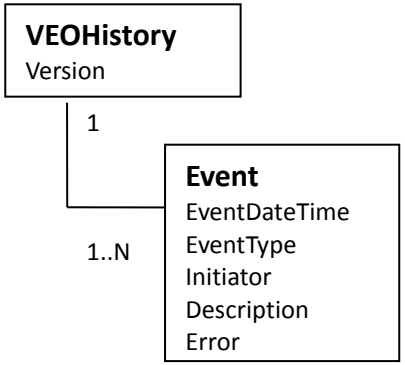


Table 3 VEOHistory.xml Structure Elements

Element Name	***Occurrences***		***Purpose***	***Value***
	Min	***Max***		
Version	1	1	Identifies the version number this VEO is conformant to	3.0
EventDateTime	1	1	States the date and time the event occurred	Date/time values are expressed in ISO 8601, normally to the nearest second. Times should be expressed in local time, not GMT or UTC.
EventType	1	1	Identifies the type of event being documented	User defined
Initiator	1	N	Identifies the person or organisation that is responsible for the event occurring	Textual string giving the name of the person or organisation. Responsibility is defined as the person who decided that the event was to take place – this might be the person who initiated the event, but it may be a more senior person who authorised it. It is preferable for a person to be identified than an organisation
Description	1	N	Contains a descriptions about the event	User defined string
Error	0	N	Contains a description of an error that was generated as a result of this event	System defined string

The XML schema for the VEOHistory.xml file is:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:vers="http://www.prov.vic.gov.au/VERS"
      targetNamespace="http://www.prov.vic.gov.au/VERS"
      elementFormDefault="qualified">

<xs:element name="VEOHistory">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Version" type="xs:string" default="3.0"/>
      <xs:element ref="vers:Event" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Event">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="EventDateTime" type="xs:string"/>
      <xs:element name="EventType" type="xs:string"/>
      <xs:element name="Initiator" type="xs:string"/>
      <xs:element name="Description" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
      <xs:element name="Error" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Step 7: Digitally sign the VEOHistory.xml file

After the VEOHistory.xml file is created it must be digitally signed at least once. Each signature and the certificates necessary to validate that signature must be placed in a VEOHistorySignature? file.

The specification for creating a VEOHistorySignature?.xml file is identical to the VEOContentSignature?.xml file (except, of course, the names of the files).

Step 8: Zipping the VEO directory

The final step in creating the VEO is to ZIP the VEO directory. The resulting ZIP file must be named with the extension '.zip' (e.g. 'TestVEO.veo.zip').

The ZIP file must be conformant with the current PKWARE APPNOTE.TXT (currently Version 6.3.3 available from http://www.pkware.com/documents/casestudies/APPNOTE.TXT)

When creating the ZIP file the following restrictions are imposed:

- the name of each ZIP entry must begin with the VEO directory name (i.e. if the VEO directory name is 'TestVEO.veo', the name of each ZIP entry must begin with 'TestVEO.veo/…' The name is correct if the VEO directory is created when manually extracting the VEO from the ZIP file).
- only the standard 'deflate' mechanism is to be used
- the encryption features are prohibited
- the digital signature features are prohibited
- the "patch data" features are prohibited.

It is expected that Zip files constructed using the default settings of the software will conform to these restrictions.

Copyright Statement

Disclaimer

The State of Victoria gives no warranty that the information in this version is correct or complete, error free or contains no omissions. The State of Victoria shall not be liable for any loss howsoever caused whether due to negligence or otherwise arising from the use of this Standard.