Version number: 2.1

Issue Date: 17 April 2024

# Introduction

## Purpose

The purpose of this procedure is to outline how to use the neoVEO toolset to create and verify Version 3 VERS encapsulated objects (VEOs) in accordance with *PROS 19/05 Create, Capture and Control Standard* technical specifications.

## Background

Public Record Office Victoria (PROV) requires permanent value digital records to be converted into a long-term format called a VEO for transfer to PROV for long term preservation. PROV also recommends that temporary value digital records with long retention periods are converted to VEOs for long-term storage and preservation.

VEOS:

* Contain digital records in an approved long-term preservation format.
* Contain contextual information (metadata) about the records.
* Are signed using digital signature technology (known as a ‘certificate’) to ensure integrity.

PROV currently supports two types of VEOs:

* **Version 2 VEOs** (created under the requirements of *PROS 99/007 Management of Electronic Records Standard*).
* **Version 3 VEOs** (created under the requirements of *PROS 19/05 Create, Capture and Control Standard*). These VEOs are more flexible than Version 2 VEOs as they allow for multi-level, hierarchical folder structures.

## Audience

This procedure has been developed for:

* Software developers in Victorian Government agencies to develop in-house tools and processes to convert their digital records into VEOs for transfer to PROV and/or long-term storage.
* Commercial software developers to create and sell records management products that can generate Version 3 VEOs.

# neoVEO toolset

## What is neoVEO?

neoVEO is the toolset developed by PROV that allow the construction, modification and analysis of Version 3 VEOs.

## What tools are in neoVEO?

The neoVEO toolset contains the following tools:

* VEOCreate. This creates VEOs from scratch using a text specification.
* VEOSign. This resigns existing VEOs.
* VEOAnalyse. This unpacks and reports on VEOs.

## How are the neoVEO tools used?

The neoVEO tools may be run as Java executables or called using an Application Programming Interface (API).

The tools must be run using Java 1.7 or later. They will not work with Java 1.6 or earlier versions.

On a computer operating a Windows operating system, the tools can be invoked using the ‘cmd.exe’ program.

## Legal

The toolset is licensed under the Creative Commons CC BY 4.0 license. This means that you have a license to do anything that you want with the toolset, provided that you:

* Acknowledge Public Record Office Victoria as the source of the toolset.
* Do not misrepresent the license or your relationship with Public Record Office Victoria.

Specifically, you may:

* Include the code from the toolset in your products, either “as is” or in a modified format.
* Use the code from the toolset as the basis of code in your toolset.

# VEOCreate tool

The VEOCreate tool creates multiple Version 3 VEOs from a control file. The control file is a text file containing multiple rows of tab separated commands. Each command builds a part of a VEO (or controls how subsequent VEOs are to be built). This class also processes the command line arguments and reads the metadata templates.

## Command line arguments

The following command line arguments must be supplied:

* **-sf <directory> the directory in which the standard VEOReadme.txt file will be found.**
* **-t <directory>** the directory in which the metadata templates will be found. See the section below for details about the metadata templates.
* **-c <file>** the control file which controls the production of VEOs. See the next section for details about the control file.

A minimal example of usage is

createVEO -c data.txt -t templates

The following command line arguments are optional:

* **-v** verbose output. By default off.
* **-d** debug mode. In this mode more logging will be generated, and the VEO directories will not be deleted after the ZIP file is created. By default off.
* **-ha <algorithm>** The hash algorithm used to protect the content files and create signatures. Valid values are: ‘SHA-1’, ‘SHA-256’, ‘SHA-384’, and ‘SHA-512’. The default is 'SHA-1'. If no –ha command line argument is present, the hash algorithm must be set in the control file. If a hash algorithm is specified in both the command line and in the control file, the control file specification overrides the command line.
* **-s <PFXfile> <password>** a PFX file containing details about the signer (particularly the private key). A PFX file is protected against reading and can only be unlocked using the password. If no -s command line argument is present, the PFX file must be specified in the control file. If PFX files are specified in both the command line and in the control file, all PFX files are used to sign the VEOs resulting in multiple signatures.
* **-o <outputDir>** the directory in which the VEOs are to be created. If not present, the VEOs will be created in the directory where the createVEO program is run.

## Control file

A control file is a text file with multiple lines. Each line contains tab separate text. The first entry on each line is the command, subsequent entries on the line are arguments to the command.

In the following list of commands, the following conventions are observed:

* Text in quotes (e.g. ‘HASH’) is entered as shown
* The text (tab) represents a tab character
* Information enclosed in ‘<’ and ‘>’ (e.g. ‘<algorithm>’) represents a string argument to the command
* Arguments enclosed in ‘[‘ and ‘]’ (e.g. (e.g. ‘[(tab) <level>]’) represent optional arguments
* Three dots (e.g. ‘[(tab) <files>…]’ indicates that the argument can be repeated separated by tab characters.

The commands:

* **'!'** A comment line. The remainder of the line is ignored.
* **'HASH' (tab) <algorithm>** Specifies the hash algorithm to use. If present, this overrides the’ –ha’ command line argument. Valid algorithms are: ‘SHA-1’, ‘SHA-256’, ‘SHA-384’, and ‘SHA-512’. HASH commands must appear before the first 'BV' command. Multiple HASH commands are allowed, but only the last is effective.
* **'PFX' (tab) <pfxFile> (tab) <password>** Specifies a PFX file and associated password to be used to digitally sign the VEO. Multiple PFX lines may be present and this results in multiple signatures being generated. If the ‘-s’ command line argument is present, both are used to generate signatures (i.e. resulting in multiple signatures). PFX commands must occur before the first BV command.
* **'VEO' (tab) <veoName> (tab) <label> (tab) <template> [(tab) <data>…] (tab)‘$$’ [(tab) <files>…]** Create a simple VEO in one command. The first argument is the VEO name (i.e. the file name of the VEO to be generated without the ‘.veo.zip’). The VEO contains one Information Object identified by the specified label (and a depth of 0). The Information Object contains one Metadata Package (identified by the template, with the substituted data), and one Information Piece that contains the specified files. If a VEO is already being constructed, a VEO command will finish the construction of the existing VEO and start a new VEO.
* **'BV' (tab) <veoName>** Begin a new VEO. The single argument is the VEO name (i.e. the file name of the VEO to be generated without the ‘.veo.zip’). If a VEO is already being constructed, a BV command will finish the construction of the existing VEO and start a new VEO.
* **'IO' (tab) <label> [(tab) <level>]** Begin a new Information Object within a VEO. The Information Object will have the specified label (which may be blank) and level. If the level is not present, it will be set to 0. If an Information Object is already being constructed, an IO command will finish the previous Information Object.
* **‘AGLS-MP’ (tab) <resourceId> Begin a new AGLS XML Metadata Package within an Information Object. Note no template is required (unlike ‘MP’ or ‘MPC’). The vers:MetadataSyntaxIdentifier is automatically set to be** “[http://www.w3.org/1999/02/22-rdf-syntax-ns#](http://www.w3.org/1999/02/22-rdf-syntax-ns#s)”**, and the vers:MetadataSchemaIdentifier to be** “[http://www.vic.gov.au/blog/wp-content/uploads/2013/11/AGLS-Victoria-2011-V4-Final-2011.pdf](about:blank)”. The AGLS/RDF preamble is automatically generated, with an **rdf:Description element containing an rdf:about attribute containing the value of the resourceID (which must be a URL). The content of this metadata package can be generated using a ME or MPC commands.** An MP, AGLS-MP, ANSZ5478-MP, XML-MP, or RDF-MP command will finish this Metadata Package and begin a new one.
* **‘ANZS5478-MP’ (tab) <resourceId> Begin a new ANZS 5478 XML Metadata Package within an Information Object. Note no template is required (unlike ‘MP’ or ‘MPC’). The vers:MetadataSyntaxIdentifier is automatically set to be** “[http://www.w3.org/1999/02/22-rdf-syntax-ns#](http://www.w3.org/1999/02/22-rdf-syntax-ns#s)”**, and the vers:MetadataSchemaIdentifier to be** “<http://www.prov.vic.gov.au/vers/schema/ANZS5478>”. The AGLS/RDF preamble is automatically generated, with an **rdf:Description element containing an rdf:about attribute containing the value of the resourceID (which must be a URL). The content of the metadata package is generated using ME, SME, EME, or MPC commands.** An MP, AGLS-MP, ANSZ5478-MP, XML-MP, or RDF-MP command will finish this Metadata Package and begin a new one.
* **‘XML-MP’ (tab) <semanticId> Begin a new generic XML Metadata Package within an Information Object. Note no template is required (unlike ‘MP’ or ‘MPC’). The vers:MetadataSyntaxIdentifier is automatically set to be “**https://www.w3.org/TR/2008/REC-xml-20081126/”**, and the vers:MetadataSyntaxIdentifier to be the specified Semantic Id string. The content of the metadata package is generated using ME, SME, EME, or MPC commands.** An MP, AGLS-MP, ANSZ5478-MP, XML-MP, or RDF-MP command will finish this Metadata Package and begin a new one.
* **‘RDF-MP’ (tab) <semanticId> (tab) <resourceID> Begin a new generic RDF Metadata Package within an Information Object. Note no template is required (unlike ‘MP’ or ‘MPC’). The vers:MetadataSyntaxIdentifier is automatically set to be “**[http://www.w3.org/1999/02/22-rdf-syntax-ns#](http://www.w3.org/1999/02/22-rdf-syntax-ns#s)”**, and the vers:MetadataSyntaxIdentifier to be the specified Semantic Id string. An rdf:Description element will automatically be added with an rdf:about attribute containing the value of the resourceID (which must be a URL). The content of the metadata package is generated using ME, SME, EME, or MPC commands.** An MP, AGLS-MP, ANSZ5478-MP, XML-MP, or RDF-MP command will finish this Metadata Package and begin a new one.
* **'MP' (tab) <template> [(tab) <subs>...]** Begin a new Metadata Package within an Information Object. The first argument is the template name, subsequent arguments are the substitutions. An MP command may be followed by MPC commands to construct a metadata package from several templates, or ME, SME & EME commands to explicitly construct metadata. An MP, AGLS-MP, ANSZ5478-MP, XML-MP, or RDF-MP command will finish this Metadata Package and begin a new one.
* **'MPC' (tab) <template> [(tab) <subs>...]** Continue a Metadata Package using another template and substitutions. An MPC must follow an MP command, but multiple MPC commands are allowed. The semantics and syntax identifiers in the template are ignored – the meaning of metadata and its syntax is set by the leading MP command.
* **‘ME’ (tab) <elementName> [(tab) <value> [(tab) <attribute>]]** Add a simple metadata element to a Metadata Package (a simple metadata element is one that has no subelements). The metadata element has the specified element name (e.g. “anzs5478:NameWords”). The metadata element has the specified value. The value is, of course, the text value of the element. If no value is present, the metadata element is empty (e.g. “<anzs5478:NameWords/>” is generated). The text in the attribute component (if present) is added as an attribute to the element (e.g. “rdf:datatype="xs:datetime"”). If you need to add an attribute to an empty element, just use a value that is a space.
* **‘SME’ (tab) <elementName> [(tab) <attribute>]** Start a complex metadata element to a Metadata Package (a complex metadata element is one that has subelements). Subsequent ME and SME commands create subelements of this element, until a EME command is encountered (note that nested complex elements can be created). The metadata element has the specified element name (e.g. “anzs5478:Name”). The text in the attribute component (if present) is added as an attribute to the element (e.g. “rdf:parseType="Resource"”).
* **‘EME’ (tab) <elementName>** End a complex metadata element in a Metadata Package (a complex metadata element is one that has subelements). The metadata element has the specified element name (e.g. “anzs5478:Name”). *Note that the program does not check that the nesting of subelements is correct; this command simply generates an end tag with the appropriate element name.*
* **'IP' (tab) [<label>] (tab) <file> [(tab) <files>...]** Add an Information Piece to the Information Object. The first (optional) argument is the label for the information piece and subsequent arguments are the content files to include in the Information Piece. At least one file must be present in the IP command. An IP command must occur after all the MP and MPC commands in an Information Object.
* **'E' (tab) <date> (tab) <event>> (tab) <initiator> [(tab) <description>...] [(tab) '$$' (tab) <error>...]** Add an event to the VEO History file. The first argument is the date/time of the event, the second a label for the type of event, the third the name of the initiator of the event. Then there are a series of arguments describing the event, and finally an option special argument ('$$') and a series of error messages. Events may occur at any point within the construction of a VEO (i.e. after a BV command).

A simple example of a control file that explicitly lists ANZS5478 metadata is:

hash SHA-1

pfx Test/signer.pfx Password

! This is a comment describing the VEO about to be constructed

BV testVEO5

AC C:/Documents/Original/S-37-6

IO Record 1

AZNS5478-MP <http://www.someIP.addr/some> URL

SME anzs5478:Record rdf:parseType="Resource"

ME anzs5478:EntityType Record rdf:datatype="xs:string"

ME anzs5478:Category Item rdf:datatype="xs:string"

SME anzs5478:Identifier rdf:parseType="Resource"

ME anzs5478:IdentifierString SomeIdentifier01 rdf:datatype="xs:string"

EME anzs5478:Identifier

(more metadata etc)

EME anzs5478:Record

IP Data S-37-6/S-37-6-Nov.docx

IO Data 2

IP Content S-37-6/S-37-6-Nov.docx S-37-6/S-37-6-Nov.docx

E 20140909 Opened Andrew Description $$ Error

E 20140910 Closed Andrew Description

## Metadata templates

The metadata template files are found in the directory specified by the ‘-t’ command line argument. Templates can be used to generate the metadata packages. Each MP or MPC command in the control file specifies a template name (e.g. 'agls'). An associated text template file (e.g. 'agls.txt') must exist in the template directory.

A template files contains the contents of metadata packages. Each template file consists of two parts:

* A single initial line that contains two URLs separated by a tab character. The first URL is identifies the meaning of the metadata (e.g. AGLS or AGRkMS), and the second URL identifies the syntax of the metadata (e.g. RDF). The following URLs are treated specially by the VEO toolkit: #todo.
* The following lines contain XML text representing the content of the template.

The XML text consists of explicit XML text, which will be included explicitly in each VEO, and substitutions. The start of each substitution is marked by '$$' and the end by '$$'. Possible substitutions are:

* **$$ date $$** - substitute the current date and time in VERS format
* **$$ [column] <x> $$** - substitute the contents of column <x>. Note that keyword 'column' is optional.

The MP/MPC commands in the control file contain the information used in the column or file substitutions. Note that the command occupies column 1, and the template name column 2. Data to be substituted consequently starts at column 3.

An example template file is:

http://prov.vic.gov.au/vers/schema/ANZS5478 http://www.w3.org/1999/02/22-rdf-syntax-ns#

<rdf:RDF xmlns:dcterms="http://purl.org/dc/terms/"

xmlns:aglsterms="http://www.agls.gov.au/agls/terms/">

<rdf:Description rdf:about="http://www.example.org/123">

<dcterms:creator><rdf:value>$$4$$</rdf:value></dcterms:creator>

<dcterms:created rdf:datatype="xsd:date">$$5$$</dcterms:created>

<dcterms:title>$$3$$</dcterms:title>

<dcterms:description>$$3$$</dcterms:description>

<aglsterms:function>$$3$$</aglsterms:function>

</rdf:Description>

</rdf:RDF>

## APIs

Two separate APIs may be used to construct VEOs. The APIs are documented using JavaDoc.

Do not mix calls to these APIs.

The Javadoc may be consulted for more details about the API.

### CreateVEOs

This provides a simple interface with just two calls. The first call configures the API (in particular, processing the command line arguments). The second call builds VEOs according to a specified control file.

### Create VEO

This provides a more powerful interface that allows direct programmatic construction of a VEO. There is a one-to-one mapping between the control file commands and the API calls.

# VEOSign tool

VEOSign performs tasks associated with resigning VEOs. It has four modes of operation:

* Verify. This verifies that the digital signatures in a VEO are valid and records the results in the VEOHistory. This command could be used before fixing a VEO to document the status of the signatures. As the VEOHistory.xml file is altered when using this command, the existing VEOHistory signatures are deleted and the VEOHistory.xml file is resigned. Note that the signatures are validated, but the validity of the certificates is not checked.
* Renew. This resigns the VEO, and records the fact of the resigning in the VEOHistory. This command could be used after altering (fixing) a VEO, or it could be used to add a new digital signature (e.g. if we decided to sign all incoming VEOs with a PROV signature). Any invalid VEOContent signatures are deleted, but valid VEOContent signatures are retained. As the VEOHistory.xml file is altered when using this command, the existing VEOHistory signatures are deleted and the VEOHistory.xml is resigned.
* Add Event. This adds an event to the VEOHistory.xml file. This command could be used to document processing. As the VEOHistory.xml file is altered when using this command, the existing VEOHistory signatures are deleted and the VEOHistory.xml is resigned.
* Create. This removes all the existing digital signatures and resigns the VEO. The VEO is otherwise unaltered (i.e. the resigning is not documented in the VEOHistory). This command could be used to create test VEOS.

The command may be used against zipped VEOs (i.e. ending in ‘.veo.zip’) or unpacked VEOs (i.e. ending in ‘.veo’).

After resigning, the VEO can be rezipped if required.

## Command line arguments

The following command line arguments must be supplied to VEOSign:

* **One of -verify, -renew, -addevent, or -create** This selects the mode of operation.
* The file names of the VEOs to resign/add events. The VEOs can either be packed (‘.veo.zip’) or unpacked (‘.veo’) files. If a packed VEO is specified, it is first unzipped before processing begins. The unzipped VEO will be created in the output directory. Unzipping will fail if the output directory already contains an unzipped VEO with the same name as the zipped VEO (e.g. ‘Sample.veo.zip’ and ‘Sample.veo’) – this is to prevent destroying sequences of alterations. If it is desired to overwrite an unpacked VEO, the -overwrite option can be used.
* **-s <PFXfile> <password>** a PFX file containing details about the signer (particularly the private key). A PFX file is protected against reading and can only be unlocked using the password. Multiple -s options can be specified, resulting in multiple digital signatures being applied.
* **-support <directory> The support directory containing the VERS configuration files. The -support command line argument is usually set in the BAT file.**

The valid command line arguments are:

* **-u <name>** A string containing the name of the user responsible for verifying or renewing the digital signatures; this string will appear in the VEOHistory documentation. If the name contains a space, it should be enclosed in double quotes (e.g. “Andrew Waugh”). If no -u option is used, the login id will be used as the name of the user. Note that this option has no effect in the -create mode as the VEOHistory is not updated.
* **-e <event>** A string containing a description of why the digital signatures were renewed or verified; this string will appear in the VEOHistory documentation. A description of the status/changes to the digital signatures is always added to the VEOHistory, so there is no need to duplicate this in the event text. If the description contains spaces (which it almost certain will), it should be enclosed in double quotes. If no -e option is used, a simple description about the changes to the digital signatures is used. Note that this option has no effect in the -create mode as the VEOHistory is not updated.
* **-o <outputDir>** the directory in which the processing is to be done, and the resulting zipped output (if requested) is created. If not specified, output directory is the directory where the resignVEO program is run.
* **-zip**. If this is specified, the altered/resigned VEO is re-zipped in the output directory. By default, the resigned VEO is NOT zipped if this would result in overwriting an existing file (e.g. if you resigned ‘SampleVEO.veo.zip’ in the current working directory, ‘-zip’ would fail as it would require overwriting the original VEO). This is to avoid overwriting the original VEO. If desired, this behaviour can be changed by the ‘-overwrite’ option. The -o option is another choice; this will create the zipped output in a different directory.
* **-overwrite**. This overwrites VEOs when zipping and unzipping VEOs. When unzipping a VEO, an existing unzipped VEO with the same name in the output directory will be overwritten. When zipping a VEO, an existing zipped VEO with the same name in the output directory will be overwritten.
* **-ha <hashAlg>**. The hash algorithm to be used in generating the signatures. If not specified, the ‘SHA-512’ algorithm is used.
* **-v** verbose output. If not specified, the program will only report on failures. If specified, the program will report on what command line options were selected, and briefly on each VEO processed.
* **-d** debug mode. In this mode more logging will be generated, and the VEO directories will not be deleted after the ZIP file is created. By default off.

A minimal example of usage is:

V3Resign -renew -support Z:/VERSCode/VERSCommon/VERSSupportFiles -s testSigner.pfx password VEO1.veo.zip

A more typical example is:

V3Resign -renew -s testSigner.pfx password -o ./resigned -zip -u “Andrew Waugh” -e “Resigned due to character errors in the vers:title metadata element” \*.veo.zip

This example assumes that the mandatory -support option is set in the BAT file, and will resign all the VEOs in the current directory, putting the results in the ‘resigned’ subdirectory.

## APIs

An API is provided to sign VEOs.

### SignVEOs

This provides a simple interface with just one call. A SignVEOs object can be constructed, and then the method resignVEOs() can then be called on the object. The constructer takes one argument: an array of strings, each of which represents an element in the command line. The resignVEOs() method executes these command line arguments.

# VEOAnalysis tool

The VEOAnalysis tool allows users to examine and test Version 3 VEOs. It performs any combination of the following tasks:

* Unpacking VEO files into VEO directories
* Testing VEOs for errors and questionable practices
* Producing a visualisation of the contents of a VEO

VEOAnalyse may work on one VEO or multiple VEOs.

## Command line arguments

The class has several operating modes which can be used together or separately. These are selected by the following command line arguments:

* **'-e'**: produce a summary of the errors and warnings found in the listed VEOs on standard out. The VEO directories are removed after execution unless the '-u' argument is specified. The default is not to produce the summary.
* **'-r'**: unpack the VEOs into VEO directories and include a full report expressed as HTML files in the VEO directory. The VEO directories remain after execution (i.e. selecting this option also selects the ‘-u’ option). The default is not to generate the report.
* **'-u'**: just unpack the VEO into VEO directories. No summary or report is produced unless one of '-e' or '-r' is present. Selecting the ‘-r’ option also selects the ‘-u’ option.

The mandatory command line arguments are:

* **'-s schemaDir'**: specifies the directory in which the XML schemas will be found (see below).

The other optional command line arguments are:

* **'-c'**: chatty mode. Report on stderr when a new VEO is commenced. This is used to show progress when processing a large number of VEOs.
* **'-v'**: verbose output. Include additional details in the report generated by the '-r' option.
* **'-d'**: debug output. Include lots more detail - mainly intended to debug problems with the program.
* **'-o directory'**. Create the VEO directories in this output directory

The VEOs to unpack, analyse, and report on are listed on the command line after the command line arguments. If a directory is included in the command line, it is searched for files ending in ‘.veo.zip’ and all such files are processed.

Simple examples of this command are:

VEOAnalysis –r –s ./schemas Eg1.veo.zip Eg2.veo.zip

VEOAnalysis –e –u –s ./schemas Eg1.veo.zip

VEOAnalysis –u –s ./schemas Eg1.veo.zip

VEOAnalysis –u –s ./schemas directory

## Setting the long term preservation formats

The VEO analysis will check to see if each Information Piece contains a valid long term preservation format. The valid formats are listed in the ‘validLTPF.txt’ file contained in the schema directory.

This file consists of a sequence of lines. Each line contains one valid file format. The format is specified by the file extension used to recognise instances of that file, including the ‘.’. Lines that begin with a ‘!’ are ignored (these are comment lines).

The contents of the standard ‘validLTPF.txt’ file is:

! Standard long term preservation formats

.txt

.pdf

.doc

.docx

.htm

.html

.css

.xml

.warc

.csv

.xls

.xlsx

.ppt

.pptx

.jpg

.jpeg

.jp2

.tif

.tiff

.mp3

.mp4

.wav

.eml

## Schema directory

The schema directory must contain the following files:

* vers2-content.xsd The XML schema for the VEOContent.xml file.
* vers2-history.xsd The XML schema for the VEOHistory.xml file
* vers2-signature.xsd The XML schema for the VEO?Signature?.xml files.
* ReportStyle.css The stylesheet used to style the VEO report HTML pages
* log4j.properties Properties to configure the logging system used by the RDF processing engine
* validLTPF.txt A list of the valid long term preservation formats accepted by the standard

Only validLTPF.txt should be edited by users.

A standard copy of this directory can be downloaded from the PROV website.

## APIs

An API is provided to unpack and analyse VEOs.

### VEOAnalysis

This provides a simple interface with just one call. The constructor is passed the command line arguments as a String array. The method test() is then called to process the VEOs.

The Javadoc may be consulted for more details about the API.

End of procedure