|  |  |
| --- | --- |
| Using the neoVEO tools | Version number: 0.1  Issue Date:  Expiry Date: 31-12-2020 |
|  |  |

|  |  |
| --- | --- |
| **Version number** | **The version number of the approved procedure** |
| 1. Approved by | 1. David Brown, Assistant Director, Records Services |
| 1. Date approved on | 1. dd/mm/yyyy |
| 1. Effective date | 1. dd/mm/yyyy |
| 1. Last amendment date | 1. dd/mm/yyyy |
| 1. Review due date | 1. dd/mm/yyyy |
| 1. Related documents | 1. Name of related document (s) |
| 1. Business owner | 1. Andrew Waugh |
| 1. Superseded documents | 1. None |
| 1. Registry file number |  |
| 1. Document control register number |  |

Draft version control, delete from final version.

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Changes Made** | **Author Making the Change** | **Date** |
| 1. 0.1 | 1. Standard format | 1. Andrew Waugh | 1. 12-03-2015 |
| 1. 0.2 |  |  |  |
| 1. 0.3 |  |  |  |
| 1. 0.4 |  |  |  |
| 1. 1.0 | 1. Final Version - Approved by Manager of [Insert functional area] |  |  |

# Introduction

## Purpose

The purpose of this procedure is to outline how to use the neoVEO tools (VEOCreate, VEOSign, and VEOAnalysis) to create and verify VERS encapsulated objects (VEOs) in accordance with the PROS 15/03.

## Scope

[Provide a brief description of the scope of this procedure. List when and to whom the procedure applies. Who is affected, which areas, which activities, where are the limits]

## Responsibilities

[In this section, include a list of the people (positions, for example, Senior Manager, Access Services) involved in the process and what their specific responsibilities are]

## Related documents

[List all related policies, processes and procedures, indicating any pre-requisites]

# The neoVEO tools

## What is neoVEO?

neoVEO is the toolset implementing the second version of the Victorian Electronic Records Strategy (VERS). VERS is a strategy that assists in the long term (> 7 years) preservation of digital objects.

## What are the neoVEO tools?

The neoVEO toolset contains tools that allow the construction, modification, and analysis of VERS Encapsulated Objects (VEOs). It 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 license. This means that you have a license to do anything that you want with the toolset, provided that you:

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

This program creates multiple 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:

* **-t <directory>** the directory in which the metadata templates and the standard VEOReadme.txt file 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.
* **-copy** If present, this argument forces content files to be copied to the VEO directory when creating the VEO. This is the slowest option, but it is the most certain to succeed.
* **-move** If present, the content files will be moved to the VEO directory. This is faster than -copy, but typically can only be performed on the same file system.
* **-link** If present, the content files will be linked to the VEO directory. This is the fastest option, but may not work on all computer systems and files. -link is the default

## 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.
* **'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. Another 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.
* **'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 is:

hash SHA-1

pfx Test/signer.pfx Password

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

BV testVEO5

AC Test/S-37-6

IO Record 1

MP agls data1 data2 data3 (etc)

MPC aglsAdd data1 data2 data3 (etc)

MPC aglsAdd data1 data2 data3 (etc)

MP agls data1 data2 data3 (etc)

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 are 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

VEOSign performs three tasks. Given a VEO directory it

* Removes any existing signature files (VEOContentSignature?.xml and VEOHistorySignature?.xml files)
* Resigns the VEO
* Zips the contents of the VEO directory to create a new VEO

The intended use of VEOSign is to allow the modification of VEOs. It is expected that a VEO will be unpacked using VEOAnalyse to create a VEO directory (or a VEO directory is created from scratch), the contents of the VEO will then be modified, and finally VEOSign invoked to regenerate the VEO.

In practice, VEOSign is very similar to VEOCreate.

## Command Line arguments

There are no mandatory arguments to VEOSign, but either the –c or the –s arguments must be supplied (i.e. you must specify a PFX file).

The valid command line arguments are:

* **-c <file>** the control file which controls the production of VEOs. See the next section for details about the control file.
* **-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.
* **-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.

Minimal example of usage are

signVEO -c data.txt

and

signVEO -s signer.pfx GK-duent

## Control File

The control file is identical in format and purpose to the control files used by VEOCreate, however the commands allowed are restricted to:

* **'!'** A comment line. The remainder of the line is ignored.
* **'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.

A simple example of a control file is:

pfx Test/signer.pfx Password

## APIs

An API is provided to sign VEOs.

### SignVEOs

This provides a simple interface with just three 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. The Javadoc may be consulted for more details about the API.

# VEOAnalysis

The VEOAnalysis program allows users to examine and test 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.

## Environment

## 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 valid 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

## The 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 upack 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.