Digital AV SDK – Record layouts & File inventory

Z-series Edition / Revision: Z07

REVISION IDENTIFIERS

Digital-AV SDK:	Z07β
SDK Document:	Z07j

The Digital-AV Software Development Kit (SDK provides the foundation for a fully working bible application), with no external dependencies. In fact, a fully operational application can be implemented with fewer than 1000 lines of code, as demonstrated by the golang sources in this SDK. As the SDK provides everything required, including data and index files. Some developers have discovered that they can be up and running in under an hour. You can easily jumpstart your development project by working with the provided golang sources, or go all in from scratch with the programming language of your choice.

The Digital-AV is entirely file based. There are zero dependencies and zero language bias (all programming languages can read files, even JavaScript and WebAssembly languages, when the files are placed on a web-server). File formats defined in this document use a consistent naming convention: the extent of each data file reveals the format and record width. The table to the right, defines the various extents of files that compose the SDK.

File Extent	File Type	Record Size
*.dx5	data	20 bytes
*.dx4	data	16 bytes
*.dx1	data	4 bytes
*.ix2	index	8 bytes
*.ix1	index	4 bytes
*.dxi	data + index	variable
*.bom	text	text
*.ascii	text	text

File extents identify the format [binary or text] and the record width. Extents of binary-formatted files begin with one of: {dx = for data; ix = for indices}. Extents that end in a numeric-digit are indicative that the files are fixed-width. The digit represents the count of 32-bit segments per record. As AV-Writ data files are available in three variants, and each variant's record width is clearly identified by its file extent: .dx5, .dx4, and .dx1. Again, that digit reveals the count of 32-bit segments per record. For example, dx4 contains four 32-bit segments per record. This multiplies out to 128 bits, which equates to a fixed record width of 16 bytes. To be clear, dx4 does not mean every field is 32-bits; it is just a convenient shorthand for depicting the record width for SDK files containing fixed-width records. An "i" in instead of a final numeric digit indicates that the binary file contains a built-in index, and that each record is variable width. Dedicated index file extents begin with .ix_ and optionally provide index-assisted access to fixed length records. Index files are either 8-bytes [*.ix2] or 4-bytes [*.ix1].

REVISION IDENTIFIERS
Digital-AV SDK: Z07β
SDK Document: Z07j

Digital-AV – Detailed description of file layouts

The weightiest data files are those named AV-Writ.*; these data files contain the stream of words for each verse of each chapter for each book. As these are not text files, the records are quite compact and some integer fields are lookups into other binary files. In essence, the entire set of binary files implement a highly efficient and compacted database of word embeddings that can be easily be manifested in RAM. The AV-Writ.* files with the widest record-width are obviously also the most information rich.

AV-Writ.dx5 (4 x uint16 + 2 x uint16 + 2 x bytes + uint16 + uint32; 160 bits)

Record 0 bits	Hebrew | Greek (4 x 16 bits)	Verse 16 bits	Caps 2 bits	WordKey 14 bits	Punc 8 bits	Transition 8 bits	PN+WordClass 16 bits	POS 32 bits
0	0x391C 0x0 0x0 0x0	0x0000	0x8___	0x0015 (in)	0x00	0xE0	0x0400	TBD
1	0x391C 0x0 0x0 0x0	0x0000	0x0___	0x0136 (the)	0x00	0x00	0x0D00	
2	0x391C 0x0 0x0 0x0	0x0000	0x0___	0x24F9 (beginning)	0x00	0x00	0x4010	
…	<< Beginning of Genesis 1 depicted above >>							
BDDB9	0x25A0 0x0 0x0 0x0		0x8___	0x0136 (the)	0x00	0xE0	0x0D00	
BDDBA	0x25A0 0x0 0x0 0x0		0x8___	0x2CB2 (revelation)	0x00	0x00	0x5010	
BDDBB	0x0978 0x0 0x0 0x0		0x0___	0x001D (of)	0x00	0x00	0x0400	
…	<< Beginning of Revelation 1 depicted above >>							
C0C91	0x1460 0x0 0x0 0x0	0x797D	0x0___	0x015C (you)	0x00	0x00	0xA2B0	
C0C92	0x0F74 0x0 0x0 0x0	0x797D	0x0___	0x0036 (all)	0xE0	0x00	0x0D00	
C0C93	0x0119 0x0 0x0 0x0	0x797D	0x8___	0x018A (amen)	0xE0	0xFx	0x8010	
<< End of Revelation 22:21 depicted above >>								

AV-Writ.dx5 begins with ***Greek & Hebrew*** word keys, which correspond to Strongs numbers in the Old & New Testament. Each English word can have up to four Strong's numbers associated with it. Strong's numbers are an integer representation of the original Hebrew/Greek words from which the English words were originally translated (Refer to the Strong's Exhaustive Concordance for additional backround information).

Hebrew | Greek word representation

Strongs #1	Strongs #2	Strongs #3	Strongs #4
1st Strongs #	2nd Strongs #	3rd Strongs #	4th Strongs #

It should be noted that while words in the Old Testament can have a maximum of four Strong's numbers representing the Hebrew associated with a single English word. The New Testament can only have a maximum of three Strong's numbers representing the Greek associated with a single English word. This is characteristic of the KJV translation, but four slots are reserved even for the greek to maintain a fixed record width across the entire bible.

Verse, is an inline index-pointer to the corresponding AV-Verse index.

The next sixteen bits can be thought of as two distinct fields: the first of those is ***Caps***: these 2-bits identify whether to apply capitolization rules to the lexical word. 0x8___ means to capitolize the first letter of the word (e.g. Lord). 0x4___ means to capitolize all letters of the the word (e.g. LORD). Clearly, in English, the first letter of the first word of a sentence is capitolized, and these bits facilitate all such capitolization rules. No bits set means that the word should be represented exactly as it appears in the lexicon. The remaining 14-bits are called the ***WordKey***, which is a lookup key for the AV-Lexicon or the AVX-Lexicon.

Punctuation Byte

Description	Bits
PUNC::clause	0xE0
PUNC::exclamatory	0x80
PUNC::interrogative	0xC0
PUNC::declarative	0xE0
PUNC::dash	0xA0
PUNC::semicolon	0x20
PUNC::comma	0x40
PUNC::colon	0x60
PUNC::possessive	0x10
PUNC::closeParen	0x0C
MODE::parenthetical	0x04
MODE::italics	0x02
MODE::Jesus	0x01

Capitalization bits and WordKey

Description	Bit Pattern (Hex)
English Word	0x3FFF (mask for lexicon lookup)
1st Letter Cap	0x8000 (example: Lord)
All Letters	0x4000 (example: LORD)

The next field is the ***Punctuation*** byte. Each word can have certain punctuation applied either as a prefix to the word, or alternatively as a suffix. An example of prefix punctuation is an open parenthesis. There are numerous examples of suffix punctuation, such as period, comma, or close parenthesis. The puncuation byte also has bits to represent italisized words in the text and even mark the words spoken by Jesus, which some bibles represent as red-colored text.

REVISION IDENTIFIERS

Digital-AV SDK:	Z07β
SDK Document:	Z07j

Person/Number (4 bits)

Description	High Nibble
Person bits	0x3 (0b--11)
Number bits	0xC (0b11--)
Indefinite	0x0 (0b--00)
1st Person	0x1 (0b--01)
2nd Person	0x2 (0b--10)
3rd Person	0x3 (0b--11)
Singular	0x4 (0b01--)
Plural	0x8 (0b10--)
WH*	0xC (0b00--)

In AV-Writ.dx5 and in AV-Writ.dx4, Person/Number (PN) is the left-most nibble of the WordClass field. PN applies to pronouns and verb casing. Whereas Modern English is not morpologically rich when it comes to verb case, Early Modern English was slightly richer with additional pronouns and verb cases for Second-Person Singular and Third-Person Singular, each distinct from the Early Modern Plural counterparts. The Digital-AV captures and preserves all these disctinct case markings. For instance, ***thy*** is second-person singular whereas Early Modern English ***you*** is always plural form of this pronoun. AV-SDK retains the markings for both person and number.

WordClass (12 bits)

NounOrPronoun	0x-03-
Noun	0x-01-
Noun: unknown gender	0x-010
Proper Noun	0x-03-
Pronoun	0x-02-
Pronoun: Neuter	0x-021
Pronoun: Masculine	0x-022
Pronoun: Non-feminine*	0x-023
Pronoun: Feminine	0x-024
Pronoun/Noun: Genitive	0x-0-8
Pronoun: Nominative	0x-06-
Pronoun: Objective	0x-0A-
Pronoun: Reflexive	0x-0E-
Pronoun: no case/gender	0x-020
Verb	0x-1--
to	0x-200
Preposition	0x-400
Interjection	0x-800
Adjective	0x-A00
Numeric	0x-B00
Conjunction	0x-C0-
Determiner	0x-D0-
Particle	0x-E00
Adverb	0x-F00

Transition bits are partially redundant with information contained within index files (*.ix2 & *.ix1). Yet these bits represent a more compact mechanism for data file traversal. However, the right-nibble is not redundant: it contains a zero-based index of the sentence of the verse which contains the token. Many verses contain only a single sentence, but some contain more. Sentence index is identified by MorphAdorner. Sentence index is always zero through fifteen (a nibble of data).

Transition (4 bits)

Description	High Nibble
EndBit	0x1-
BeginningOfVerse	0x2-
EndOfVerse	0x3-
BeginningOfChapter	0x6-
EndOfChapter	0x7-
BeginningOfBook	0xE-
EndOfBook	0xF-

* ***his*** is used ambiguously in the Authorized Version for third-person-singular pronouns. ***his*** is either masculine or neuter (***its*** appears just once in the sacred text). Therefore, ***his*** can neither be uniformly marked as masculine, nor neuter. Instead, we mark the genitive pronoun ***his*** as non-feminine.

There are two additional trimmed down versions of the AV-Writ files which contain subsets of the first data file described above. These can be used for more memory constrained implementations or utilized where the additional data fields are not needed.

AV-Writ.dx4 (4 x uint16 + 2 x uint16 + 2 x bytes + uint16 = 128 bits)

Record 0 bits	Hebrew | Greek (4 x 16 bits)	Verse 16 bits	Caps 2 bits	WordKey 14 bits	Punc 8 bits	Transition 8 bits	PN|WordClass 16 bits
0	0x391C 0x0 0x0 0x0	0x0000	0x8___	0x0015 (in)	0x00	0xE0	0x0400
1	0x391C 0x0 0x0 0x0	0x0000	0x0___	0x0136 (the)	0x00	0x00	0x0D00
2	0x391C 0x0 0x0 0x0	0x0000	0x0___	0x24F9 (beginning)	0x00	0x00	0x4010
…	<< Beginning of Genesis 1 depicted above >>						
C0C91	0x1460 0x0 0x0 0x0	0x797D	0x0___	0x015C (you)	0x00	0x00	0xA2B0
C0C92	0x0F74 0x0 0x0 0x0	0x797D	0x0___	0x0036 (all)	0xE0	0x00	0x0D00
C0C93	0x0119 0x0 0x0 0x0	0x797D	0x8___	0x018A (amen)	0xE0	0xF0	0x8010
<< End of Revelation 22:21 depicted above >>							

AV-Writ.dx1 (uint16 + 2 x bytes = 32 bits)

Record 0 bits	Caps 2 bits	WordKey 14 bits	Punc 8 bits	Transitions 8 bits
0	0x8___	0x0015 (in)	0x00	0xE0
1	0x0___	0x0136 (the)	0x00	0x00
2	0x0___	0x24F9 (beginning)	0x00	0x00
…	<< Beginning of Genesis 1 depicted above >>			
C0C91	0x0___	0x015C (you)	0x00	0x00
C0C92	0x0___	0x0036 (all)	0xE0	0x00
C0C93	0x8___	0x018A (amen)	0xE0	0xF0
<< End of Revelation 22:21 depicted above >>				

The ascii variations of SDK files are intended to be informative, and are not considered to be core components of the inventory. Non-optimal performance would be achieved if a choice were made to develop against the ascii files instead of the binary formats. Yet, they are provided in the SDK to illuminate the formats of similarly organized binary files.

AV-Book.ix2 (UInt16 + 2 x byte + 32 ASCII characters = 64 bits)

Record 0 bits	Chapter Index 16 bits	Book Number 8 bits	Chapter Count 8 bits	Book Name 15 bytes	MatchCount 8 bits	Abbr 32bit	Abbr 32bit	Abbr 32bit	Abbr 32bit
0	0x000	1	50	Genesis	2	Gn			
1	0x032	2	40	Exodus	2				
2	0x05A	3	27	Leviticus	2				
…									
65	0x4A4	66	22	Revelation	2	Rn			

REVISION IDENTIFIERS
Digital-AV SDK: Z07β
SDK Document: Z07j

AV-Chapter.ix2 (UInt32 + 2 × UInt16 = 64 bits)

Record 0 bits	Bible Index 32 bits	Verse Index 16 bits	Word Count 16 bits
0x000 (genesis:1)	0x00000	0x0000	0x31D
0x001 (genesis:2)	0x0031D	0x001F	0x278
0x002 (genesis:3)	0x00595	0x0038	0x2B7
	. . .		
0x4A2 (revelation:20)	0xC0769	0x797B	0x1DD
0x4A3 (revelation:21)	0xC0A56	0x797C	0x2ED
0x4A4 (revelation:21)	0xC0C93	0x797D	0x23D

AV-Verse.ix1 (4 × byte = 32 bits)

Record# 0 bytes	Book, Chapter, Verse, Words 32 bits: BB:CC:VV:WordCnt	
0x0000	0x0101010A	In the beginning …
0x0001	0x0101021D	And the Earth …
0x0002	0x0101030A	And God said …
	...	
0x797B	0x4216152C	And if any man … are written in this book.
0x797C	0x42161510	He which testifieth … Even so, come, Lord Jesus.
0x797D	0x4216150C	The grace of our Lord … be with you all. Amen

AV-Lexicon.dxi (variable length; replaceable by AVX-Lexicon.dxi or AVX-Lexicon.FB)

Rec#	WordSize 2 bytes (16 bits)	WordCnt 2 bytes (16 bits)	Null-separated 8-bit character arrays WordCnt*2 bytes
0	1	3	'a\0i\0o\0' [keys = 1,2,3]
1	2	40	'ah\0ai\0am\0 ... ye\0' [keys = 4,5,6, ... ,353]
2	3	311	'abi\0act\0add\0 ... zur\0' [keys = 354,355,356, ... ,1440]
	…		
12565	18	2	'jonathelemrechokim\0maher-shalal-hash-baz\0' [keys = 12565, 12566]
12566	0	12567	(total)

AV-Lexicon.dx is the most compact of the two variant lexicons. As such, it less lexical information than AVX-Lexicon.

A usage example for AV-Lexicon.dx is provided in avtext.go. A usage example for AVX-Lexicon is provided in avx.go.

AVX-Lexicon.dxi (data and index combined: variable length records)

Rec#	Search uint16 [ ] 5-bit chars	Display uint16 [ ] 5-bit chars	Modern uint16 [ ] 5-bit chars	Entities uint16	Size uint16	1 POS Uint32	2 POS Uint32	3 POS Uint32	• • •	N POS Uint32
0	0x0001	0x0000	0x0000		N=5	0x00000094	0x00000036	0x0000000A		0x00000279
1	0x0009	0x0000	0x0000		N=4	0x01074F9C				
2	0x000F	0x0000	01E8		N=4	0x000002A8				
…										
3	Thou		you							
…										
12566										

Entities = {Hitchcock=0x8000, men=0x1, women=0x2, tribes=0x4, cities=0x8, rivers=0x10, mountains=0x20, animals=0x40, gemstones=0x80, measurements=0x100}

The Implicit key on both variations of the Lexicon is merely the order of the records. The key is one-based. Accordingly, the record-key for "a" is 1. Record-keys on AV-Lexicon.dxi, AVX-Lexicon.dxi are 100% compatible. A word of caution about AV-Lexicon.dxi: text entries are NOT null terminated, but are simple ascii-encoded byte arrays, prefixed by a size-byte. Additional encoding on AV-Lexicon.dxi exists on all words that contain hyphens; hypens are not stored as ordinary ascii characters; instead, the byte that proceeds the hyphen sets the negative bit on the byte (i.e. 0x80). When this bit is set on a letter, a hyphen should be inserted after that character (a remnant from when code was expected to run within a megabyte of RAM or less). In any case, the letter-portion of the byte can be extracted with a mask of 0x7F. Only the AV-Lexicon.dxi data file exhibits this legacy encoding. AVX-Lexicon.dxi uses a different text compaction mechanism. In AVX-Lexicon, all strings are 5-bit encoded. Each 16-bit segment contains three 5-bit characters (the leading 0x8000 indicates that an additional 16-bit segment is required). The first 16-bit segment is null-padded on the left (each overflow segment always contains three characters). AVX-Lexicon differs not only in format, but provides additional content as well. Specifically, it provides modern orthographic representation for archaic words, and an array of one or more Part-of-Speech (POS) fields associated with the word. The POS field list captures every POS combination encountered in the biblical text.

AV-PartOfSpeech.dx2

PartOfSpeech 32 bits	Count 16 bits	WordClass 16 bits
		be
		went
		run
...		

The original AVX-Lemma file dates back to the 2017 Edition of the SDK. The old version obtained Lemmata from the NLTK Python library. This updated AV-Lemma file obtains Lemmata from the MorphAdorner Java server (MorphAdorner also performs all of the POS tagging). Incidentally, each Lemma ordinarily maps to multiple English words or lexemes, (e.g. The lemma 'be' corresponds to 'are', 'were', 'is', 'art', and 'be'). Moreover, words like 'run' can function both as a verb and a noun. Accordingly, Part-of-Speech needs to be considered when accessing the lemma utilizing AVX-Lemma for looking up the lemma for a word.

More information:

REVISION IDENTIFIERS
Digital-AV SDK: Z07β
SDK Document: Z07j

AV-Lemma.dxi (variable length)

Part-of-Speech (POS) 32 bits	wordkey 16 bits	Lemma uint16 [ ] 5-bit chars
		be
		went
		run
…		
0		

AV-WordClass.dxi (data and index combined: variable length records)

WordClass 16 bits	Width 16 bits	1st POS 32 bits	2nd POS 32 bits	3rd POS 32 bits	. . .	Nth POS 32 bits
0x0001	N=5	0x00000094	0x00000036	0x0000000A		0x00000279
0x0009	N=4	0x01074F9C				
0x000F	N=4	0x000002A8				
. . .						

AV-Names.dxi (data and index combined: variable length records)

NameLen 1 byte	Name Len bytes	Count 1 byte	1st Len 1 byte	1st Meaning Len bytes	2nd Len 1 byte	2nd Meaning Len bytes	3rd Len 1 byte	3rd Meaning Len bytes	⋮	Nth Len 1 byte	Nth Meaning 32 bits
0x05	Aaron	0x03	0x09	a teacher	0x05	lofty	0x14	mountain of…			
0x07	Abaddon	0x01	0x0D	the destroyer							
0x07	Abagtha	0x01	0x18	father of the…							
. . .											

avtext.go (golang source code)

AVtext.go implements a web-server (HTTP server) that provides the entire text of the AV bible using simple semantics. As the web-server is not hardened, it should be placed behind a reverse-proxy if exposed to the open Internet. This is a common pattern and Caddy, a more general-purpose web-server, can be easily configured as a reverse-proxy.

There are a couple of URLs used for testing and validation. They also illustrate how avtext.go can be extended:

- http://localhost:2121/
- http://localhost:2121/help
- http://localhost:2121/validate

Example of GoLang source in operation may be available at avbible.net:

https://avbible.net/av/
(the web-site above also utilizes Caddy as a reverse-proxy for HTTPS)

The / endpoint simply reports the release number of the avtext.go web-server component. The /help endpoint provides primitive information about the web-service. /help can be easily replaced by developer. The /validate endpoint reports on the validity of data files in accordance with the bom (The "bom", or bill of materials, is described in the section labelled AV-Inventory.bom later in this document. In addition to the administrative URL's described above, here is a list of the foundational endpoints that provide the core functionality of avtext.go:

1. http://localhost:2121/genesis
2. http://localhost:2121/genesis/1
3. http://localhost:2121/gen/1?sessionID
4. http://localhost:2121/rev/22?sessionID=day&amen
5. http://localhost:2121/rev/22?sessionID=$FFFFFFFFFFFF
6. http://localhost:2121/css/sessionID.css

All of these endpoints can be summarized as one of two types: getting the chapter of a book, or getting a CSS stylesheet. When no chapter is provided, chapter 1 is always implied. When no session identifier is provided, the resulting chapter request is decorated with the baseline stylesheet, named /css/AV-Stylesheet.css. When a session identifier is provided, the session number dictates the name of the CSS file that will decorate the chapter request. Moreover, avtext.go can compile information into a CSS stylesheet. When a request is made for Genesis using the URL depicted in #3 above, a stylesheet becomes linked in the response to a stylesheet with the URL depicted in #8 above. A web-browser will make an immediate subsequent request to get the stylesheet. If /css/sessionID.css does not exists, avtext.go will automatically compile a file named /css/sessionID.avspec. Similarly, but easier to understand in #4 above, the URL would generate CSS which would highlight the words **day** and **amen**. In order to maintain optimal performance, session identifiers are non-volatile. In order to overwrite a *.css files and/or *.avspec files, they must be manually deleted beforehand.

URL form #3 and #5 are discussed under the description of the *.avspec format.

avx.go (golang source code)

avx.go implements a web-server (HTTP server) that provides the entire text of the AV bible with several AVX extensions, but still uses simple semantics. As the web-server is not hardened, it should be placed behind a reverse-proxy if exposed to the open Internet. This is a common pattern and Caddy, a more general-purpose web-server, can be easily configured as a reverse-proxy.

There are a couple of URLs used for testing and validation. They also illustrate how avx.go can be extended:

- http://localhost:2121/
- http://localhost:2121/help
- http://localhost:2121/validate

Example of GoLang source in operation may be available at avbible.net:

https://avbible.net/avx/
(the web-site above also utilizes Caddy as a reverse-proxy for HTTPS)

The / endpoint simply reports the release number of the optional avx.go web-server component. The /help endpoint provides primitive information about the web-service. /help can be easily replaced by developer. The /validate endpoint reports on the validity of data files in accordance with the bom (The "bom", or bill of materials, is described in the section labelled AVX-Inventory.bom later in this document). In addition to the administrative URL's described above, here is a list of the foundational endpoints that provide the core functionality of avx.go:

1. http://localhost:2121/avx/genesis
2. http://localhost:2121/avx/genesis/1
3. http://localhost:2121/avx/gen/1?sessionID
4. http://localhost:2121/avx/rev/22?sessionID=day&amen
5. http://localhost:2121/avx/rev/22?sessionID=$FFFFFFFFFFFF
6. http://localhost:2121/avx/css/sessionID.css

All of these endpoints can be summarized as one of two types: getting the chapter of a book, or getting a CSS stylesheet. When no chapter is provided, chapter 1 is always implied. When no session identifier is provided, the resulting chapter request is decorated with the baseline stylesheet, named /css/AV-Stylesheet.css. When a session identifier is provided, the session number dictates the name of the CSS file that will decorate the chapter request. Moreover, avx.go can compile information into a CSS stylesheet. When a request is made for Genesis using the URL depicted in #3 above, a stylesheet becomes linked in the response to a stylesheet with the URL depicted in #8 above. A web-browser will make an immediate subsequent request to get the stylesheet. If /css/sessionID.css does not exists, avx.go will automatically compile a file named /css/sessionID.avspec. Similarly, but easier to understand in #4 above, the URL would generate CSS which would highlight the words **day** and **amen**. In order to maintain optimal performance, session identifiers are non-volatile. In order to overwrite a *.css files and/or *.avspec files, they must be manually deleted beforehand.

URL form #3 and #5 are discussed under the description of the *.avspec format.

*.avspec file format

WordKey Count UInt16	Array of UInt16	
0xnnnn	0xnnnn Count of WordKeys is followed by WordKey list [corresponds to AV-Lexicon]	
BookChapter UInt16	**Verse Count byte** *(matching verses)*	**Array of byte**
0xbbcc	0xkk	0xkk Count of matching verses is followed by an array of Verse numbers
…		
0xbbcc	0xjj	0xjj Count of matching verses is followed by an array of Verse numbers
0x0000		

AVtext.go software ignores everything after the first record above. Only that first record defines the CSS file. And that first line is expanded word-for-word into highlights for each supplied wordkey. A slight variation here is that Strong's numbers will eventually also support highlighting. To highlight Strong's numbers, set the 0x8000 bit for Hebrew and the 0x4000 bit for Greek. The URL form that was depicted with this syntax, sessionID=$FFFFFFFFFFFF, is primarily intended for testing. Here, the hex digits that follow the dollar sign ($) are expected to be expansions of the format described above (No record separators, just a representation of the raw bytes described above, in Big-Endian order).

AV-Stylesheet.css (text file containing CSS for avtext.go; optional)

This standard-format CSS stylesheet should be included when avtext.go is utilized in your development. This optional stylesheet is included in the SDK, but it can be customized in any way by the web designer. However, the web designer should realize that any references in the CSS to image files will result in 404 errors unless support is explicitly added to avtext.go by your development team. Finally, avtext.go always links chapter output to the AV-Stylesheet.css stylesheet, even when a *.avspec derived stylesheet is also specified.

AV-Inventory.bom & AVX-Inventory.bom (text file which identifies core inventory; optional)

This is a text file that identifies the release and the delivered files for any given release. "bom" stands for bill of materials. For each artifact of the SDK, the bom lists each filename along with its corresponding MD5. The avtext.go server implements a validation function that will read this file and report if the MD5's agree with the SDK files on disk. This way corruption can be detected and as a revision can be validated in an automated fashion. Each Plate revision is released with a bom that should be deployed with the SDK. By doing this, streamlined plate revision checks are straightforward. MD5's are calculated both on core SDK files, and on optional components, but not upon the bom itself. Please note that the bom is not updated on every Alpha/Beta release, but can be updated manually using avtext.go. After downloading AV-Inventory.bom, it is recommended that you copy each bom to a name that will not be overwritten. For each revision, it is recommended that you would copy AV-Inventory.bom to another location (e.g. AV-Inventory.Z07) along with the documentation associated with that release [this document may change over time].

OVERALL PROJECT STATUS:

It's an exciting time at AV Text Ministries, and if you want to lend a hand, let us know your technical skills and interests and we can help jumpstart you onto the team. Currently, AV Text Ministries is 100% volunteer, so if you don't just have passion about the mission as your raw motivation, it might not be the best fit.

Finally, on the non-technical side of things, we would certainly welcome a ministry sponsor that would want to place AV Text Ministries under the banner of their own local church ministry. Check out http://ReceivedText.com, http://Digital-AV.org, and http://avtext.org to discover our overall vision.

HOW THE DIGITAL-AV "PLATES" ARE AUTHORED:

Initially, various publicly available KJV texts were parsed and dutifully compared (comparing scripture with scripture [1 Corinthians 2:13]). That work produced the freeware program, AV-1995 for Windows; it was written in Delphi/Pascal and was maintained through 2011. In 2008, the initial Digital-AV SDK was conceived and produced, harvesting much of the inner workings of AV-2008, utilizing RemObjects Oxygene/Pascal as a development platform and releasing it as open source. Later, the 2011 Edition was "compiled" based upon the 2008 Edition of the SDK. Subsequently, the 2017/2018 Editions were "compiled" based upon the 2011 Edition. The Z07 revision of the SDK was "compiled" entirely using the latest 2018 [i728] as a baseline. C# is nor the programming language of the SDK compile, as the pascal sources were finally completely retired (replaced by C# sources) in 2018. The Z-series SDK compiler leveraged the MorpAdorner server, which is written in Java 1.6 (http://morphadorner.northwestern.edu/morphadorner/) and a custom Python/Flask server for NLTK (http://www.nltk.org). Both servers are accessed via REST calls to local servers running on the program author's network on localhost. POS tags are acquired from the MorphAdorner server. Lemmatization is also primarily acquired from the MorphAdorner server, with the NLTK Flask server utilized only if the MorphAdorner fails to return a lemma (We are unsure if this is happening at this point; a later SDK may use the NLTK for quality control and/or verification).

Digital AV SDK – Record layouts & File inventory

Z-series Edition / Revision: Z07

REVISION IDENTIFIERS

Digital-AV SDK:	Z07β
SDK Document:	Z07j

LICENSE REQUIREMENT:

- In order to comply with the MIT-style open source license, please include AV-License.txt with your distribution of any file identified in this SDK. The text of that file as of 2020 is provided also at the bottom of this page.

All SDK artifacts are on github.com:

https://github.com/kwonus/Digital-AV

IMPROVEMENTS & CAVEATS:

- Fundamental SDK format has stabilized and is substantially similar to the 2017 and 2018 editions. However, the 2020 Edition has substantial changes to the AVX extensions within the SDK. Specifically, the sqlite lexicon has been eliminated and the binary format of AVX-Lexicon is also substantially different. A structured lexicon is now available as a Flat Buffers data file. Additionally, another field was added to AV-Writ that provides a precise Part-of-Speech representation for each word. Finally, the .dx? & .ix? extents now reflect a count of 32-bit segments instead of a count of 16-bit segments. AV-Lemma is new in the 2020 SDK also.
- Part-of-speech (POS) bits were introduced into most of AV-Writ.* files in the HA29 release,. As of the R07 release, POS bits have been substantially revised as the SDK now uses MorphAdorner for part-of-speech marking instead of NLTK (NLTK doesn't recognize archaic verbs and pronouns, whereas MorphAdorner does).

ADDITIONAL RELEASE NOTES:

#1 The "Z-series" edition of the SDK introduced an updated revision number format in 2020. Digital-AV revision numbers now use a three-digit character sequence, plus an optional suffix. All revision numbers bow begin with the letter **Z**. The next two characters represent year and month of the revision. The character sequence is **Z*ym*** where the first letter is always **Z**, indicating that this is the "Z-series" edition of the SDK (distinguishing it from older/legacy SDK editions); ***y*** represents the year, and ***m*** represents the month of the release. ***y*** encodes the year as a single base-36 digit; For example, (*y* = 0) represents 2020; (*y* = 9) represents 2029; (*y* = A) represents 2030; (*y* = K) represents 2040; (*y* = U) represents 2050. With respect to months, digits 1 through 9 are as expected; (*m* = A) is October; (*m* = B) is November; and (*m* = C) is December. An optional one-digit suffix may also be used. If the suffix is a Greek letter (α or β), then this identifies an alpha or beta release of the SDK. Otherwise a suffix identifies the discrete date of the release, encoded in base-36; the 1st is 1, the 31st is u. A date-specific suffix is ordinarily reserved only for documentation. Yet, source code tagging in github can also include the date suffix.

#2 Two revision numbers exist: the Digital-AV SDK revision (aka, the "plate" revision) is the most significant set of files. Not that all files in this SDK are required to produce working bible software. Incidentally, the sample source code provided in avtext.go implements a minimal SDK artifacts-usage set, while still providing access to the entire AV Bible text.

#3 Many of the binary files also have corresponding text files with an .ascii extent. These files are not provided for runtime execution. Instead, they should be considered as ancillary documentation to shed light, in painstaking detail, on the corresponding binary files.

#4 The SDK is comprises mostly binary files, prefixed either with AV- or AVX-. The files prefixed with AV- are core elements of the SDK and represent pure representations of the original Authorized Version texts. The AVX- files are considered extensions to the core SDK and are entirely optional components of the SDK.