Digital AV SDK – Record layouts & Content inventory

Release Version: 5.1

Digital-AV Software Development Kit provides the foundation for a fully working bible application, with no external dependencies. The SDK provides everything, including data with indexes. Be up and running in under an hour! Easily jumpstart your development by leveraging sources in the foundational C# reference implementation.

Filename: AVX-Omega-5104.data
Total-Length: 19,651,823
Content-Hash: 35A36F16A07D72C98F6031BD785C7AD9 *(of AVX-Omega-5104.data // from AVX-Omega-5104.md5)*

The description of all content, artifact by artifact, is provided in the following pages of this document.

Digital AV SDK – Record layouts & Content inventory

Release Version: 5.1

VERSION IDENTIFIERS

Digital-AV SDK: Ω5104

SDK Document: Ω5104

Directory Content (48 bytes per record)

Content Label char[16]	Content Offset uint32	Content Length uint32	Record Length uint32	Record Count uint32	Content Hash uint64 [2]
Directory	0	432	48	9	[0x0000000000000000, 0x0000000000005104]
Book	432	3,216	48	67	[0x8D91A88B86A3F567, 0x952B6E7DF85AAB6F]
Chapter	3,648	7,134	6	1,189	[0x29D5C0D1AFF79C95, 0xBE3A964891126FBE]
Written	10,782	18,951,624	24	789,651	[0x9C0E15FD2919E7D9, 0x7D88EA922F517619]
Lexicon	18,962,406	247,066	0	12,567	[0x4D440402AC14153A, 0x46D1A2A14B77945D]
Lemmata	19,209,472	182,344	0	15,171	[0xB64F907ABC54470F, 0x2D227D8AC5703E33]
OOV-Lemmata	19,391,816	7,754	0	771	[0xADEA45027082EC56, 0xEA59B079EF94C96F]
Names	19,399,570	60,727	0	2,470	[0xB7885CB9C8F0293A, 0x3845818BD5A4DCEC]
Phonetic	19,460,297	191,526	0	13,337	[0x78A31E514E3292EA, 0xF0C7ED99D199CDFF]

The Digital-AV SDK (AV SDK) is entirely file based, with zero dependencies and zero language bias (all programming languages can read a file). The Ω-Series SDK is derived from the earlier Z-Series SDK. The main difference is the Ω-Series uses a single content file, beginning with a content directory as depicted above (The Z-Series releases have multiples files and a separate inventory file similar to the directory). The directory identifies the content sections for easy deserialization. The first field is the label, followed by an offset, and a length (in bytes). It also includes record length: zero (0) indicates that the record is variable length; non-zero length means that the record is fixed length. Record count, and content hash [using MD5] are the final fields for each directory item.

The file format defined in this document pertains to the Ω-Series releases. Both Ω and Z series formats are similar, but not identical. Consequently, some formats that have been revised in the Ω-Series do not have corresponding revisions in the Z-Series SDK specification. In fact, the Z-Series SDK is deprecated. While the older Z-Series SDK is solid, new development should utilize this newer SDK.

VERSION IDENTIFIERS
Digital-AV SDK: Ω5104
SDK Document: Ω5104

Written Content (24 bytes per record)

Record # *0 bits*	Hebrew or Greek *4 x uint16*	B:C:V:W *4 x byte*	Caps *0xC 2bits*	Word Key *14 bits*	Punc *byte*	Transition *byte*	POS-Bits *uint16*	NUPOS *uint32*	OOV *0xC 2bits*	Lemma *14 bits*
0	0x391C 0 0 0	1:1:1:10	0x8	0x0015 (in)	0x00	0xE8	0x00E0	0x40080470	0x0	0x0015
1	0x391C 0 0 0	1:1:1:9	0x0	0x0136 (the)	0x00	0x00	0x0D00	0x00000094	0x0	0x0136
2	0x391C 0 0 0	1:1:1:8	0x0	0x24F9 (beginning)	0x00	0x00	0x4010	0x000001DC	0x0	0x24F9
…	<< Beginning of Genesis 1 depicted above >>									
0xBDDB9	0x25A0 0 0 0	66:22:21:35	0x8	0x0136 (the)	0x00	0xE0	0x0D00	0x00000094	0x0	0x0136
0xBDDBA	0x25A0 0 0 0	66:22:21:34	0x8	0x2CB2 (revelation)	0x00	0x00	0x4010	0x000001DC	0x0	0x2CB2
0xBDDBB	0x0978 0 0 0	66:22:21:33	0x0	0x001D (of)	0x00	0x00	0x0400	0x80004206	0x0	0x001D
…	<< Beginning of Revelation 1 depicted above >>									
0xC0C91	0x1460 0 0 0	66:22:21:3	0x0	0x015C (you)	0x00	0x00	0x20C0	0x00083BBD	0x0	0x015C
0xC0C92	0x0F74 0 0 0	66:22:21:2	0x0	0x0036 (all)	0xE0	0x04	0x0D00	0x00000004	0x0	0x0036
0xC0C93	0x0119 0 0 0	66:22:21:1	0x8	0x018A (amen)	0xE0	0xFC	0x8000	0x8000550E	0x0	0x018A
<< End of Revelation 22:21 depicted above >>										

The most substantial content contained in the directory is that which is Written. It represents the stream of words for each verse of each chapter of each book of the KJV bible. These are not text files. Therefore, they are quite compact. Several fields are index lookups into other SDK content. In short, the collective content manifests an efficient database of word embeddings that resides compactly in RAM.

The first field of Written content contains Strong's numbers[1]. These are a numeric representation of the original Hebrew/Greek words from which the sacred text was originally translated.

Hebrew | Greek

Strong's #1	Strong's #2	Strong's #3	Strong's #4
1st numeric	2nd numeric	3rd numeric	4th Strongs #

[1] Refer to the Strong's Exhaustive Concordance for additional backround information. The Digital-AV has at most, four Strong's numbers per English word in the Old Testament. By contrast, in the New Testament, there are at most three Strong's numbers per English word. To maintain a fixed length record format, four slots allotted for both testaments.

VERSION IDENTIFIERS
Digital-AV SDK: Ω5104
SDK Document: Ω5104

The Ω-series releases eliminate the AV-Verse index and place that information directly in the Written content records instead. Indexing into Written from Chapter provides verse coordinates and word counts. Navigating to subsequent verses is accomplished via the word-count for the verse. The first word always contains the word count of the verse; each subsequent word contains a countdown until one (i.e. that last word of the verse is marked with a *:*:*:1)

Word Key & Capitalization Field

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

The next sixteen bits can be thought of as two distinct fields: the first 2 bits, ***Caps***, 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. When no bits are set, this indicates that the word should be represented exactly as it appears in the lexicon. The remaining 14-bits are referred to as the ***Word Key*** (a lookup key into the Lexicon). The next field is the ***Punctuation*** byte. Each word can be preceded by punctuation (e.g. an open parenthesis). More often, punctuation follows the word. The ***Punctuation*** byte also contains possible ***Decoration***. Decoration includes italisized words, and words spoken by Jesus, which some bibles represent as red-colored text. The field is entirely bitwise and many forms of punctuation and decoration can simultaneouslt apply to a single word in the text.

Punctuation & Decoration

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

Transition bits are a composition of Verse-Transitions and Segment-Markers. These represent a compact mechanism for data file traversal, obviating the need for leveraging additional index files. The five left-most bits mark book, chapter, and verse transitions. The three right-most bits mark linguistic segmentation [sentence and/or phrase] boundaries. These boundaries are based upon verse transitions and punctuation.

Transition byte has two distinct parts

Verse Transition

Description	5-bits
EndBit	0x10
BeginningOfVerse	0x20
EndOfVerse	0x30
BeginningOfChapter	0x60
EndOfChapter	0x70
BeginningOfBook	0xE0
EndOfBook	0xF0
BeginningOfBible	0xE8
EndOfBible	0xF8

Segment Transition

Description	3-bits
HardSegmentEnd	0x04
CoreSegmentEnd	0x02
SoftSegmentEnd	0x01
RealSegmentEnd	0x06

Hard Segments: `. ? !`
Core Segments: `:`
Real Segments: `. ? ! :`
Soft Segments: `, ; ( ) --`

POS (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

POS-Bits is a sixteen-bit field with the left-most nibble representing Person Number (PN). PN applies to pronouns and verb casing. Early Modern English was richer than our English today, with additional pronouns and verb cases for Second-Person-Singular and Third-Person-Singular. The Digital-AV captures and preserves all such case markings. For instance, ***thy*** is second-person singular whereas Early Modern English ***you*** is always plural form of this pronoun. The SDK encodes the markings for both person and number using the binary representation depicted in the table to the right. Similarly, remaining bits provide part-of-speech markers. The remaining twelve bits provide bitwise POS information for this word in its context. The table to the left shows the meaning of the various bits.

Person/Num (4 bits)

Description	Left-Most 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--- (0b11--)

* ***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.

NUPOS

There is an additional POS32 field that has much greater fidelity on the part-of-speech for the word. NUPOS is a five-bit encoding of a human readable string. The encoded field can be expanded into a NUPOS string.

Lemma (the Lemma word has two distinct parts)

The final field of the Written content record is the key for lemmatization of the word. Similar to Word-Key, the first two bits (0xC000) provide augmented information. If the 0x8000 bit is set for the Lemma, it should be looked up in the OOV-Lemmata. When the 0x8000 bit is not set, the lemma should be looked up in the standard lexicon. When the bit indicates a lookup in the lexicon is required, the value should be masked by 0x3FFF. The mask is required, as the 0x4000 bit is an indicator that the word should not be modernized. The GetModern() method in the library consults this bit to squelch modernization when appropriate. Normally, when looking up the modernized word in the lexicon, we fetch the string in the modern column of the lexicon. However, there is at least one entry in the lexicon where the original KJV word should not be modernized in all situations. The modernization-quelch bit appropriately controls squelching.

OOV & the modernization-squelch bit

Description	Bit Pattern (Hex)
Lexicon Lookup	0x3FFF (mask for lexicon lookup)
OOV Lookup	0xBFFF (mask for lookup in OOV-Lemmata)
Quelching modernization	0x4000 (modernization-squelch bit)

As of the Ω4220 revision, only a single word in the lexicon requires modernization to be squelched. That word is “art”. “Art” is the form of “to be” that agrees with the pronoun “thou”. When modernized, “art” becomes “are” and “thou” becomes “you”. However, in Acts 17:29, “art” is used as a noun. In that instance, it would be wrong to modernize it into a verb. Using the quelch bit allows us to maintain a simple bi-text lexicon. Again, to date, Acts 17:29 is the only known word in the lexicon that needs the squelching-bit to be set.

Book Content (48 bytes)

Book Number *byte*	Chapter Count *byte*	Chapter Index *uint16*	Writ Count *uint16*	Writ Index *uint32*	Book Name *16 bytes* (utf8)	Abbreviations (utf8) a2 a3 a4 *(12 bytes)*	Abbreviations (utf8) Alternates *(10 bytes)*
0	0	0	0	0x4220	Omega 4.2.20----	42-o42-Ω42	----------
1	50	0	38262	0	Genesis---------	Ge-Gen-Gen--	Gn--------
2	40	50	32685	38262	Exodus----------	Ex-Exo-Exod-	----------
3	27	90	24541	70947	Leviticus-------	Le-Lev-Lev--	Lv--------
66	22	1167	11995	777656	Revelation------	Re-Rev-Rev--	----------

Book content provides indicies into Chapter content, Written content. It also provides chapter-counts and word-counts (for each of the sixty-six books of the bible). It reserves a fixed sixteen-byte field for the book-name, a fixed nine-byte field (2+3+4) for 2-character, 3-character, and 4-character abbreviations. The remaining nine bytes are a comma-delimited list of any additional alternate abbreviations.

The dashes (-) represent zero ('\0'). The nine byte field above, namely "a2 a3 a4" comprises 2-character, 3-character, and 4-character abbreviations. Note that the format now contains 67 records instead of 66. The zeroth record contains metadata about the revision and conveniently makes record #1 correspond to book #1.

Chapter Content (6 bytes)

Record # 0 bits	Writ Index Uint16	Writ Count uint16	Book Num byte	Verse Count byte
0x000 (genesis:1)	0	797	1	31
0x001 (genesis:2)	797	632	1	25
0x002 (genesis:3)	1429	695	1	24
	. . .			
0x4A2 (revelation:20)	10196	477	66	15
0x4A3 (revelation:21)	10673	749	66	27
0x4A4 (revelation:22)	11422	573	66	21

NOTE:

In the Omega SDK, verse metadata is easily accomplished by looking-up the Written content record indexed via the Writ-Index of the Chapter content record. As every Written content record has full coordinates in the B:C:V:W field, we can quickly get the Book-Number, Chapter-Number, Verse-Number, and Word-Count for the first verse of any chapter. Once we have the first verse, walking the rest of the chapter is trivial.

VERSION IDENTIFIERS
Digital-AV SDK: Ω5104
SDK Document: Ω5104

Lemmata content originally appeared in the 2017 Edition of the SDK. Earlier, Lemmata was obtained from the NLTK Python library. Now, Lemmata are obtained from the MorphAdorner Java service (Additionally, MorphAdorner performs all POS-tagging; This Java reference is performed during SDK compilation; a Java runtime dependency is not required). Incidentally, each Lemma can map to more than one English words or lexemes, (e.g. 'be' is the lemma of 'are', 'were', 'is', 'art', 'wast', and 'be'). Interestingly, many words, for example 'run', are not even constrained to a single part-of-speech. Consequently, Lemmata lookup requires a POS tag. Successful lookups into Lemmata produces a set of WordKeys or OOVKeys. When a Lemma is OOV (i.e. not found in Lexicon), find it via the OOV-Lemmata table).

Lemmata Content (variable length records)

Part-of-Speech (POS32): uint32	Word Key uint16	POS-Bits: uint16	Count uint16	Lemmata Array Uint16[] (Word or OOV keys)
0x00000036	0x0001 (a)	0x0F00	1	0x0001
0x00000094	0x0001 (a)	0x0D00	1	0x0001
0x80004206	0x0001 (a)	0x0400	1	0x0001
0x01074F9C	0x0002 (i)	0x4080	1	0x0002
…				
0x00003A1C	0x027A (elim)	0x4030	1	0x027A
0x000001DD	0x027B (elms)	0x8010	1	0x8304 (OOV: elm)
…				
0xFFFFFFFF				

OOV-Lemmata Content (variable length records)

OOV Key uint16	OOV Word Length+1 bytes
0x8301	aid\0
…	
0x8F01	covenantbreaker\0

OOV (composition by example)

OOV Marker 1 bits	OOV Length 7 bits	OOV Index byte
0x8___	0x_3__	0x__01

(binary of 0x8301 is b1000001100000001)

The Lexicon provides both original and modern orthographic representations for each lexeme identified in the Written content; and it includes a search representation, stripping out all hyphens. What follows are an array of associated Part-of-Speech (NUPOS representation). These are 5-bit encoded value. A reference implementation for decoding this POS value into a human readable POS string can be found in the github repo.

Lexicon Content (variable length records)

Rec # (0 bits)	Entities uint16	Size uint16	POS[0] uint32	POS[1] uint32	POS[2] uint32	• • •	POS[n-1] uint32	Search char [ ]	Display char[ ]	Modern char [ ]	
0	0xFFFF	n=2	12567	0x5104							metadata
1	0x0000	n=4	0x00000094	0x00000036	0x0000000A		0x80004206	a			Entities = { } dt, av, j, pp-f
2	0x0000	n=3	0x01074F9C	0x0000000A	0x01073F9C			i			Entities = { } pns11, j, pno11
3	0x0000	n=1	0x000002A8					o		oh	{ } oh
…											
366	0x8009	n=2	0x00003A1C	0x000740FC				adam			Entities = {Man, City} np1, npg1
…											
1311	0x0000	n=2	0x01073FBC	0x0000000A				thou		you	Entities = { } pns21, j
…											
12567	0x0000	n=1	0x0000000A					Mahershal alhashbaz	Maher-shalal-hash-baz		Entities = { } j

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

NOTE:

Note that the lexicon contains a zeroth-record, making lex-key equal to record-index (a Word-Key of zero is invalid).

Additional notes about Part-of-Speech in Digital-AV

Both the POS-Bits field and the NUPOS field are found in Written content. And both represent Part-of-Speech, in different, but related manners. As POS-Bits operates bitwise, it is easier to make programmatic determinations based upon that field. The NUPOS field is 5-bit encoded into a UInt32. Decoding the 32-bit value into a string (see citation below). POS [NUPOS] tagging was extracted from Morph-Adorner (also cited below). POS-Bits is derived both from the MorphAdorner tag and innate knowledge in the Digital-AV compiler of pronouns and morphology. The NUPOS field is an encoded human-readable string. An earlier version of the SDK contained a HashMap, mapping each POS32 value into a collection of POS12 values. However, that file was deemed incomplete and has been eliminated from the SDK. That mapping might be useful, but is easily inferred from Written content. AV-Lexicon contains only POS32 references, and no POS12 references.

In short, the POS-Bits field is more granular and has a bitwise representation. Contrariwise, the NUPOS fields have far more fidelity, but require decoding to expose their string representation.

For more information, see:

- https://github.com/kwonus/Digital-AV/blob/master/z-series/Part-of-Speech-for-Digital-AV.pdf
- https://github.com/kwonus/blob/master/blueprint-blue/Blueprint-Blue-Lib/FiveBitEncoding.cs

  [*method signature:* **string DecodePOS(Uint32 encoding)**]

Names Content (variable length records)

uint16	1st Meaning	Delimiter	2nd Meaning	Delimiter	3rd Meaning	Delimiter	• • •
Lexicon WordKey for ‘Aaron’ (0x05AC)	a teacher	|	lofty	|	mountain of…	\0	
Lexicon WordKey for ‘Abaddon’ (0x14DC)	the destroyer	\0					
Lexicon WordKey for ‘Abagtha’ (0x14DD)	father of the…	\0					
. . .							

Names Content is a binary representation of “Hitchcock's Bible Names Dictionary”, authored by Roswell D. Hitchcock in 1869. The difference here is that it is integrated by indexing with the word-key found in AV-Lexicon.

Phonetic Content (variable length records; one record for all lexicon entries and all OOV lemmata)

WordKey uint16	1st IPA Variant	Delimiter	2nd IPA Variant	Delimiter	3rd IPA Variant	Delimiter	• • •
Lexicon WordKey for ‘a’ (0x0001)	ˈeɪ	/	ə	\0			
. . .							
Lexicon WordKey for ‘Aaron’ (0x05AC)	ˈɛɹən	\0					
. . .							
Lexicon WordKey for ‘baptist’ (0x1578)	ˈbæptəst	/	ˈbæptɪst	\0			
. . .							
Lexicon WordKey for ‘receive’ (0x1B02)	ɹəˈsiv	/	ɹiˈsiv	/	ɹɪˈsiv	\0	
. . .							
OOV key for ‘aid’ (0x8301)	ˈeɪd	\0					
. . .							
OOV key for ‘covenantbreaker’ (0x8F01)	ˈkəvənəntˈbɹeɪkɚ	\0					
. . .							

American phonetic representations are provided for all entries in the lexicon and all OOV lemmata in the Phonetic Content payload. Most phonetic representations are provided in IPA. Constructed IPA representations (lexical items that are not found in the file named en_US.txt[2]) may contain NUPhone[3] representations instead of IPA.

[2] En_US.txt can be found at https://github.com/open-dict-data/ipa-dict/blob/master/data/en_US.txt. This file is used to look up strings and substrings to generate IPA.
[3] NUPhone representation is described at https://github.com/kwonus/NUPhone/blob/main/NUPhone.md

OVERALL PROJECT STATUS:

It's an exciting time at AV Text Ministries, and if you want to lend a hand. Let me know your technical skills and interests and I can help jumpstart you into the process. 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, I would certainly welcome a ministry sponsor that would want to place AV Text Ministries under the banner of their own local church ministry. Check http://avtext.org to discover my 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 C++ and Delphi/Pascal and was maintained until 2011. In 2008, the initial Digital-AV SDK was conceived and released. It harvested much of the inner workings of AV-2008. A transition from Delphi to RemObjects Oxygene began in 2008 and continued until 2015 (Oxygene is Delphi-like Pascal that compiles to .NET Windows). While a fun expirement, it became clear that the Pascal and C++ sources needed to be replaced with C#.

2015 marked the beginning of making C# the primary programming language of the SDK-compiler. AV-Bible-2021 was re-released as a ground-up C#/WPF application using the 2018 Digital-AV SDK. Today, there is not a trace remaining of the Pascal sources.

Incidentally, Digital-AV is entirely language agnostic. There is a single binary file with an index that can be processed from ANY language. That said, a full reference implementation for the SDK is available. The reference implementation is written in C# 12 and is found in AVXLib under foundations in the Digital-AV repo on github. That library emerged from a multi-year development effort that culminated in 2024.

The SDK itself is generated using an SDK-compiler. The SDK-compiler is written in C#, but it hits REST services: it uses MorpAdorner[4] (written in Java 1.6), along with the NUPOS [5]tag-set; it also uses NLTK[6] (Python), but NLTK is only consulted when MorphAdorner encounters a word out of its vocabulary. Java and Python dependencies are not exhibited in the delivered SDK. Neither is there a C# dependency in the SDK itself. These are merely dependencies of the SDK compiler/generator.

The Z-Series introduced new features, but retained much of the format of 2018 SDK. Initially, the Omega SDK versions were compiled using the latest Z-Series SDK as its baseline. However, all recent Omega SDK releases utilize the previous Omega release as their baselines. Omega 4.2 is built from Omega 3.9. Major.Minor version numbers are Y.M where Y = year modula 10; and M is the hex representation of the month (1=January, 2=February, … C=December)

Only the http://github/AV-Text/AVX repository has the Omega SDK generation code. All other generated artifacts can be found at:
https://github.com/kwonus/Digital-AV

To be clear, only the generated artifacts [found at Digital-AV repo] are required for building applications. Additionally, AVX-Framework provides a tested foundation if your application is a .NET application. Various repos compose the AVX-Framework. Those are peer repositories of Digital-AV and can be found at:
https://github.com/kwonus/

NOTE: The Omega releases were inspired by the simplicity of utilizing Google FlatBuffers: why mess with a bunch of files when we can mess with just one? The Z-Series assets are no longer being maintained. However, they can still be utilized as much of the content is cut from the same cloth.

---

[4] http://morphadorner.northwestern.edu/morphadorner/
[5] https://github.com/kwonus/Digital-AV/blob/master/z-series/Part-of-Speech-for-Digital-AV.pdf
[6] http://www.nltk.org

Digital AV SDK – Record layouts & Content inventory

Release Version: 5.1

VERSION IDENTIFIERS

Digital-AV SDK: Ω5104
SDK Document: Ω5104

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. That license is provided also at the bottom of this page.

All SDK artifacts are on github.com:

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

IMPROVEMENTS & CAVEATS:

- Underlying SDK formats have stabilized in the Z-series and Ω-series editions.
- The huge difference between the Z-series and Ω-series editions is that Omega edition utilizes a single file for deserialization.
- The Omega release introduced revised Written, Book, and Chapter content (which differed from the earlier Z-Series SDK)
- The Omega release also omit Verse content. Instead, verse coordinates are provided directly in the Written content. In conjunction with Chapter content, the functionality provided by the older Verse Content block is easy to navigate. Page-8 elaborates how these two indexes work in tandem.
- The name of the serialization file is **AVX-Omega-YMDD.data**; and **AVX-Omega-YMDD.md5** contains a hash of the data file.
- For the most part, the Omega releases share most of the same formats as the earlier Z-Series SDK. Ω4220+ is the recommended SDK for future development.
- Hashing values in Directory & total-verse-counts for each Book were incorrect in the 3.2 release. This necessitated the 3.5 release. Only Directory & Book content were revised in the **Ω**3507 revision. Other issues were found in the Ω3507 revision. Directory & Book content were further revised in the **Ω**3911 revision [Anyone still referencing **Ω**3507 is urged to rebase development efforts on **Ω**3911, **Ω**4220, or **Ω**5104].
- The **Ω**4220 revision adds the modernization-squelch-bit to the Lemma field. See page-6 for details. Version identifiers and content hashes have also been updated.
- The Lexicon was updated in the **Ω**5104. This does not impact AV/KJX renderings, but additional AVX/modernizations are found in the latest releases. Refer to the link here for additional details about this update: https://github.com/kwonus/AVBible/blob/omega/Help/language.md

ADDITIONAL RELEASE NOTES:

#1 Digital-AV revision numbers use a four-digit character sequence, (alpha/beta releases use three-digits and an alpha or beta Greek letter suffix/subscript). The most recent revision numbers begin with **Ω**. The next two characters represent year and month of the revision. The character sequence is either ***Ωymdd*** or ***Ωymdd$_x$*** ; ***y*** represents the year, and ***m*** represents the month. ***y*** encodes the year as a single base-36 digit; For example, (*y*=0) represents 2020; (*y* == 3) represents 2023; (*y* == 5) represents 2025; (*y* == A) represents 2030; (*y* == F) represents 2035; (y == Z) represents 2055. With respect to months, digits 1 through 9 are as expected; (*m* == A) is October; (*m* == B) is November; and (*m* == C) is December. A two-digit day of the month follows. If a Greek letter subscript appears in the version number (α or β), then this indicates an alpha or beta release.

#2 Two revision numbers are provided in the specification: The Digital-AV SDK revision (aka, the "plate" revision) is the most significant as it identifies the actual serialization file. There is also a distinct and separate revision number for the document itself. Finally, when appendices are included, those also have distinct revision numbers.

#3 Introduced in the **Ω**3905 revision is phonetic representations for all modern lexical items and all OOV lemmata. This can be used as a lookup for any keyed string provided in the Lexicon Content and in the OOV-Lemmata Content

Digital AV SDK – C# Foundational Support

Release Version: 5.1

VERSION IDENTIFIER
C# Support: Ω5104

With the Ω-series SDK, we open just a single binary file to extract all SDK content. C# Foundational support is substantially unchanged from the 4.2 release, albeit it now utilizes 5.1 data files and MD5 hashes.

Written content is segmented into 66 different arrays and placed in the Book index/content (one slice for each book of the bible).

The code has undergone acceptance testing and fortifies the rock-solid foundation upon which it is built. The SDK has decades of usage and most of the recent developments are improvements in packaging/deployment.

An older C# project was AVXText. That project was more monolithic and the command interpreter was neither decoupled from bible content search, nor from bible content rendering. Moreover, the older AVXText project has been abandoned.

This framework, written in C# 12 and compiled with .NET 8, is fully decoupled using purpose-specific libraries. These discrete libraries mitigate issues encountered by the earlier implementations. Details can be found in the AVX-Framework documentation:
https://github.com/kwonus/AV-Engine

Incidentally, the new C# foundational support can leverage other pieces of the AVX-Framework for a broad set of compatible features:

- Searching (AV-Search)
- Text Similarity [sounds-alike] term matching (NUPhone)
- Command Interpreter (Quelle / Pinshot-Blue / Blueprint-Blue)
- Foundational AV/AVX Bible (AVXLib)

The C# Foundational library is dependency-free and can be found at:
https://github.com/kwonus/Digital-AV/blob/master/omega/foundations/csharp/AVXLib