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

***BREAKING NEWS:*** Direct support for Rust, C++, and FlatBuffers is debuting in the Z31 revision. See the addendums at the bottom of this document for status of each of these endeavors. I expect great progress in Q1 2023.

|  |  |  |  |
| --- | --- | --- | --- |
| **File**  **Extent** | **File**  **Type** | **Record**  **Type** | **Content**  **Type** |
| \*.dx | data | fixed length | binary |
| \*.ix | index | fixed length | binary |
| \*.dxi | data + index | variable length | binary |
| \*.bom | MD5 checksums | newline delimited | text |
| \*.ascii | informational | newline delimited | text |

The base Digital-AV SDK (AVSDK) is entirely file based. There are zero dependencies and zero language bias (all programming languages can read files). File formats defined in this document use a consistent naming convention: the extent of each data file reveals the content and record type. The table to the right, defines the various extents of files that compose the SDK.

|  |  |  |
| --- | --- | --- |
| **AV-Writ**  **variants** | **Size in bits** | **Size in**  **bytes** |
| AV-Writ.dx | 176 bits | 22 bytes |
| AV-Writ-128.dx | 12byte | 16 bytes |
| AV-Writ-32.dx | 32 bits | 4 bytes |

File extents identify the format/file type. Fixed-width data files have \*.dx file extents, while fixed-width index files have \*.ix file extents. A third file-type combines the data and index into a single file (and those binary files are always variable length). The \*.bom file contains MD5 hashes which can be used at runtime to verify the file conforms to the release. The \*.ascii files provide additional information for the developer and are not expected to be deployed to the end-user.

AV-Writ.dx file has three variants to handle disparate memory and/or system constraints. While the file formats are detailed later in this document, the table to the right is provided to summarize the records widths for each variant. (only a single AV-Writ\* file need be deployed with your application). It is up to the developer to weigh the footprint versus features in that decision

# **Digital-AV – Detailed description of file layouts**

# The weightiest data files are those named AV-Writ\*.dx; these files contain a stream of words for each verse of each chapter for each book. As these are not text files, the records are compact, with some integer fields being an index lookup into another SDK file. In essence, the entire set of binary files implement an efficient database of word embeddings, designed to be compactly manifested in RAM. The AV-Writ\*.dx files with the widest records are obviously also the most feature rich.

# AV-Writ.dx (22 bytes per record)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Record #**  ***0 bits*** | | **Hebrew | Greek**  ***4 x uint16*** | **Verse**  ***uint16*** | **Caps**  ***2 bits*** | **WordKey**  ***14 bits*** | **Punc**  ***byte*** | **Transition**  ***byte*** | **PN+WClass**  ***uint16*** | **POS**  ***uint32*** | **Lemma**  ***uint16*** |
| 0 | | 0x391C 0 0 0 | 0 | 0x8\_\_\_ | 0x0015 (in) | 0x00 | 0xE8 | 0x00E0 | 0x40080470 | 0x0015 |
| 1 | | 0x391C 0 0 0 | 0 | 0x0\_\_\_ | 0x0136 (the) | 0x00 | 0x00 | 0x0D00 | 0x00000094 | 0x0136 |
| 2 | | 0x391C 0 0 0 | 0 | 0x0\_\_\_ | 0x24F9 (beginning) | 0x00 | 0x00 | 0x4010 | 0x000001DC | 0x24F9 |
| … | | << Beginning of Genesis 1 depicted above >> | | | | | | | |  |
| 0xBDDB9 | | 0x25A0 0 0 0 | ­30698 | 0x8\_\_\_ | 0x0136 (the) | 0x00 | 0xE0 | 0x0D00 | 0x00000094 | 0x0136 |
| 0xBDDBA | | 0x25A0 0 0 0 | 30698 | 0x8\_\_\_ | 0x2CB2 (revelation) | 0x00 | 0x00 | 0x4010 | 0x000001DC | 0x2CB2 |
| 0xBDDBB | | 0x0978 0 0 0 | 30698 | 0x0\_\_\_ | 0x001D (of) | 0x00 | 0x00 | 0x0400 | 0x80004206 | 0x001D |
| … | | << Beginning of Revelation 1 depicted above >> | | | | | | | |  |
| 0xC0C91 | | 0x1460 0 0 0 | 31101 | 0x0\_\_\_ | 0x015C (you) | 0x00 | 0x00 | 0x20C0 | 0x00083BBD | 0x015C |
| 0xC0C92 | | 0x0F74 0 0 0 | 31101 | 0x0\_\_\_ | 0x0036 (all) | 0xE0 | 0x04 | 0x0D00 | 0x00000004 | 0x0036 |
| 0xC0C93 | | 0x0119 0 0 0 | 31101 | 0x8\_\_\_ | 0x018A (amen) | 0xE0 | 0xFC | 0x8000 | 0x8000550E | 0x018A |
|  | << End of Revelation 22:21 depicted above >> | | | | | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Strongs #1** | **Strongs #2** | **Strongs #3** | **Strongs #4** |
| 1st Strongs # | 2nd Strongs # | 3rd Strongs # | 4th Strongs # |

AV-Writ.dx begins with ***Greek & Hebrew*** Strong’s 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. While words in the Old Testament can have a maximum of four Strong’s numbers associated with a single English word.

Hebrew | Greek encodings representation

The New Testament can only have a maximum of three Strong’s numbers 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 entire bible. For more information on Strongs numbers, refer to the Strong's Exhaustive Concordance for additional backround information. Also note that ***Verse*** is an inline index-pointer to the corresponding AV-Verse index.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

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

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. 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 called the ***WordKey*** (a lookup key for the AV-Lexicon). Incidentally, the lookup key is equally compatible with the Lexicons found in the older 2018 SDK (Revision #K817).

Capitalization bits and WordKey

Punctuation Byte

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.

Within AV-Writ.dx and in AV-Writ-128.dx, 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 encodes the markings for both person and number using the binary representation depicted in the table to the left.

Person/Number (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--- (0b00--) |

|  |  |
| --- | --- |
| 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 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]

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

boundaries. In this edition of the SDK, these bondaries are interpretted based upon a combination of verse transitions and punctuation.

Verse Transitions

**\*** ***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.

WordClass (12 bits)

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

AV-Writ-128.dx (16 bytes per record)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Record**  **0 bits** | **Hebrew | Greek**  **4 x uint16** | **Verse**  **uint16** | **Caps**  **2 bits** | **WordKey**  **14 bits** | **Punc**  **byte** | **Transition**  **byte** | **PN|WordClass**  **uint16** |
| 0 | 0x391C 0x0 0x0 0x0 | 0x0000 | 0x8\_\_\_ | 0x0015 (in) | 0x00 | 0xEF | 0x00E0 |
| 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 | 0x20C0 |
| C0C92 | 0x0F74 0x0 0x0 0x0 | 0x797D | 0x0\_\_\_ | 0x0036 (all) | 0xE0 | 0x06 | 0x0D00 |
| C0C93 | 0x0119 0x0 0x0 0x0 | 0x797D | 0x8\_\_\_ | 0x018A (amen) | 0xE0 | 0xFE | 0x8000 |
|  | << End of Revelation 22:21 depicted above >> | | | | | | |

AV-Writ-32.dx (4 bytes per record)

Segment Markers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Record**  **0 bits** | **Caps**  **2 bits** | **WordKey**  **14 bits** | **Punc**  **byte** | **Transitions**  **byte** |
| 0 | 0x8\_\_\_ | 0x0015 (in) | 0x00 | 0xEF |
| 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 | 0x06 |
| C0C93 | 0x8\_\_\_ | 0x018A (amen) | 0xE0 | 0xFE |
|  | << End of Revelation 22:21 depicted above >> | | | |

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

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

AV-Book index provides indicies into AV-Chapter, AV-Verse, and AV-Writ, and corresponding chapter-counts, verse-counts, and word-counts (for each of the sixty-six books of the bible). It reserves a fixed sixteen bytes for the book-name, a fixed nine bytes (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.

# AV-Book.ix (50 bytes)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Record #**  ***0 bits*** | **Book**  **Number**  ***byte*** | **Chapter**  **Count**  ***byte*** | **Chapter**  **Index**  ***uint16*** | **Verse Count *uint16*** | **Verse Index *uint16*** | **Writ Count *uint32*** | **Writ**  **Index**  ***uint32*** | **Book**  **Name**  ***16 bytes* (utf8)** | **Abbreviations (utf8)** | | |
| **a2 a3 a4**  ***(9 bytes)*** | **Alternates**  ***(9 bytes)*** | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 31 | Z31.9----------- | -- --- ---- | | --------- |
| 1 | 1 | 50 | 0 | 1533 | 0 | 38262 | 0 | Genesis--------- | Ge Gen Gen- | | Gn------- |
| 2 | 2 | 40 | 50 | 1213 | 1533 | 32685 | 38262 | Exodus---------- | Ex Exo Exod | | --------- |
| 3 | 3 | 27 | 90 | 859 | 2746 | 24541 | 70947 | Leviticus------- | Le Lev Lev- | | Lv------- |
| … |  | | | | | | | | | | |
| 66 | 66 | 22 | 1167 | 404 | 30698 | 11995 | 777656 | Revelation------ | Re Rev ---- | |  |

The dashes (-) above represent zero ('\0'). In the "a2 a3 a4" column, these nine bytes contain a 2-character abbreviation, a 3-character abbreviation, and a 4-character abbreviation, The AV-Book.ix file has an updated format, beginning with the Z31 revision. Note that the newer format now contains 67 records instead of 66. The zeroth record contains metadata about the revision and makes record #1 correspond to book #1. The previous AV-Book.ix has been renamed AV-Book-Z14.ix. Either file is fully compatible with all Z-series revisions. Weighing in with only 1,238 additional bytes, the newer file likely results in fewer ancillary lookups.

# AV-Book-Z14.ix (32 bytes) *Compatible with earlier SDK releases, including Z14*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Record #**  **0 bits** | **Book Number**  **byte** | **Chapter Count**  **byte** | **Chapter Index**  **Uint16** | **Book Name**  **16 bytes (utf8)** | **Book Abbreviations**  **12 bytes (utf8)** |
| 0 | 1 | 50 | 0 | Genesis | Ge |
| 1 | 2 | 40 | 50 | Exodus | Ex |
| 2 | 3 | 27 | 90 | Leviticus | Le |
| … |  | | | | |
| 65 | 66 | 22 | 1167 | Revelation | Re |

|  |
| --- |
| AV-Chapter.ix (10 bytes) |

nibble)

|  |
| --- |
| AV-Chapter-Z14.ix (8 bytes) |

nibble)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Record #**  **0 bits** | **Writ**  **Index**  **uint32** | **Writ Count**  **uint16** | **Verse Index**  **uint16** | **Verse Count**  **uint16** |
| 0x000  (genesis:1) | 0 | 797 | 0 | 31 |
| 0x001  (genesis:2) | 797 | 632 | 31 | 25 |
| 0x002  (genesis:3) | 1429 | 695 | 56 | 24 |
|  | **. . .** | | |  |
| 0x4A2  (revelation:20) | 787852 | 477 | 31039 | 15 |
| 0x4A3  (revelation:21) | 788329 | 749 | 31054 | 27 |
| 0x4A4  (revelation:22) | 789078 | 573 | 31081 | 21 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Record #**  **0 bits** | **Writ**  **Index**  **uint32** | **Verse Index**  **uint16** | **Word Count**  **uint16** |
| 0x000  (genesis:1) | 0 | 0 | 797 |
| 0x001  (genesis:2) | 797 | 31 | 632 |
| 0x002  (genesis:3) | 1429 | 56 | 695 |
|  | **. . .** | | |
| 0x4A2  (revelation:20) | 787852 | 31039 | 477 |
| 0x4A3  (revelation:21) | 788329 | 31054 | 749 |
| 0x4A4  (revelation:22) | 789078 | 31081 | 573 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Record#**  **0 bytes** | **Book, Chapter, Verse, Words**  **4 bytes: BB:CC:VV:WordCnt** |  |
|  | 0x0000 | 1:1:1:10 | In the beginning … |
|  | 0x0001 | 1:1:2:29 | And the Earth … |
|  | 0x0002 | 1:1:3:11 | And God said … |
|  | … | |
|  | 0x797B | 66:22:19:44 | And if any man … are written in this book. |
|  | 0x797C | 66:22:20:16 | He which testifieth … Even so, come, Lord Jesus. |
|  | 0x797D | 66:22:21:12 | The grace of our Lord … be with you all. Amen |

***NOTE:***

AV-Chapter.ix differs from earlier revisions, as it how includes verse count and an altered column order than the earlier Z14 Revision.

|  |
| --- |
| AV-Verse.ix (4 bytes) |

nibble)

AV-Chapter contains one extra data field than its Z14 counterpart, but both files are fully compatible with all Z-series revisions.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Part-of-Speech**  **uint32** | **Word Key**  **uint16** | | **Word Class**  **uint16** | **Count**  **uint16** | **Lemma 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 Key**  **uint16** | **OOV Word**  **Length+1 bytes** |
| 0x8301 | aid\0 |
| … |  |
| 0x8F01 | covenantbreaker\0 |
| … |  |

The AV-Lemma file originally appeared in the 2017 Edition of the SDK. The original version obtained Lemmata from the NLTK Python library. Now Lemmata are obtained 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. ‘be’ is the lemma of ‘are’, ‘were’, ‘is’, ‘art’, ‘wast’, and ‘be’). Moreover, words like ‘run’ can function both as a verb and a noun. Part-of-Speech needs to be considered when accessing the lemma utilizing AV-Lemma for looking up the lemma for a word. Lemmas contains a list of WordKeys and/or OOVKeys. When a Lemma is OOV[[1]](#footnote-1), it cannot be found in the Lexicon, but it can be found in the OOV table.

|  |
| --- |
| AV-Lemma.dxi (variable length) |

nibble)

# AV-Lemma-OOV.dxi (lookup for OOV lemmas)

# OOV (composition by example)

|  |  |  |
| --- | --- | --- |
| **OOV Marker**  **1 bits** | **OOV Length**  **7 bits** | **OOV**  **Index**  **byte** |
| 0x8\_\_\_ | 0x\_7\_\_ | 0x\_\_01 |
| (binary of 0x8301 is b1000001100000001) | | |

# AV-Lexicon provides both original and modern orthographic representations for each lexeme identified in AV-Writ. It also includes a search-version of the lexeme that strips out all hyphens. Next, an array each Part-of-Speech (POS) associated with that lexical item is identified by a encoded value using 5-bit character encoding. A reference implementation of to decode a Uint32 value into a human readable POS string can be found in the github repo.

# AV-Lexicon.dxi (data and index combined: 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=3 | 12567 | 31 | 9 |  |  | |  |  |  | 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 |  |  |  |  | | Mahershalalhashbaz | 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:*** AV-Lexicon.Z31 differs from Z14 revision: it inserts a zeroth-record, making lex-key equal to record-index. It also differs by omitting the marker/final record after record #12567, as did the Z14 Revision. Otherwise, they are identical.

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **WordClass**  **uint16** | **Width**  **uint16** | **1st POS**  **uint32** | **2nd POS**  **uint32** | **3rd POS**  **uint32** | **⚫ ⚫ ⚫** | **Nth POS**  **uint32** |
| 0x0010 | N=4 | 0x4000394E | 0x00003950 | 0x40075AC7 |  | 0x40075ACE | n-jn, njp, n-vvg, n-vvn |
| 0x00E0 | N=1 | 0x01074F9C |  |  |  |  | p-acp |
| 0x0100 | N=29 | 0x00005842 | 0x000B0893 | 0x00005847 |  | 0x00005ADA | Vbb, vbds, vbg, … vvz |
| ... | | | | | | |  |

# It should be noted that both the 16-bit WordClass field and each POS field contains part-of-speech information, but the 16-bit WordClass field is more granular and has a bitwise representation. Contrariwise, the encoded 32-bit POS fields have far more fidelity, but require decoding.

# 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/AVXText/blob/master/FiveBitEncoding.cs> [ *method signature:* **string DecodePOS(Uint32 encoding)** ]

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **WordKey**  **uint16** | **1st Meaning** | **Delimiter** | **2nd Meaning** | **Delimiter** | **3rd Meaning** | **Delimiter** | **⚫ ⚫ ⚫** |
| AVLexicon WordKey for Aaron | a teacher | | | lofty | | | mountain of… | \0 |  |
| AVLexicon WordKey for Abaddon | the destroyer | \0 |  |  |  |  |  |
| AVLexicon WordKey for Abagtha | father of the… | \0 |  |  |  |  |  |
| ... | | | | | | | |

# avx.go (golang source code)

avx.go implements a web-server (HTTP server) that provides the entire text of the AV bible utilizing AVX extensions, but still uses simple semantics. Version numbers for source code are respective of the SDK Document revision numbers. The first release of avx.go, which had been updated to the z-series SDK, was the Z081 golang source-code revision.

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

<https://avbible.net/avx/>

(the web-site above also utilizes NGINX as a reverse-proxy for HTTPS)

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

* <http://localhost:2121/>

*NOTES:*

1. 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; Apache httpd, NGINX, Caddy, or IIS can easily be configured to serve as a reverse-proxy.
2. URL form #3 and #5 are discussed under the description of the \*.avspec format

* <http://localhost:2121/help>
* <http://localhost:2121/validate>

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 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 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=$FFFFFFFFFFFFF>
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. Avx.go uses Z08 edition.

# \*.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 |  | |

avx.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=$FFFFFFFFFFFFF, is primarily intended for testing. Here, the hex digits that follow the dollar sign ($) 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 avx.go; optional)

This standard-format CSS stylesheet should be included when avx.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 avx.go by your development team. Finally, avx.go always links chapter output to the AV-Stylesheet.css stylesheet, even when a \*.avspec derived stylesheet is also specified.

# AV-Inventory-Z31.bom (text file which identifies core inventory)

This is an ascii text file that provides a bill-of-materials for the delivered files. For each file of the SDK, the bom contains a line item for the artifact. Each line has three fields, separated by whitespace:

1. Name of the file
2. MD5 hash (hexadecimal representation) of the file
3. Record Length (decimal representation // uint32) of the file // 0 for variable-width files
4. Record Count (decimal representation // uint32) of the file
5. Size in bytes (decimal representation // uint32) of the file

The avx.go server implements a validation function, using an older bom; it reads the bom and reports inconsistencies. To avoid malicious attack, utilization of the bom is highly recommended, but not required. It helps mitigate corruption, both intentional and unintentional.

# AV-Inventory-Z31.md5 (new with the Z31 release)

This ascii text file contains the MD5 of AV-Inventory-Z31.bom. For utmost security, utilized this MD5 to check the validity of AV-Inventory-Z31.bom, in addition to checking the validity of each file utilized at runtime from the bom.

**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. We are embarking on brand-new support for Rust. 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 <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 until the AV-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, AV-2011 was “compiled” using AV-2008 as a baseline. Subsequently, the 2017/2018 Editions were “compiled” using AV-2011 as a baseline. The Z07 revision of the SDK were baselined from AV-2018 edition using the K817 revision. C# is now the programming language of the SDK compiler; and the ancient pascal sources were finally retired (replaced by C# sources) in 2018. The SDK-compiler uses MorpAdorner[[2]](#footnote-2) (written in Java 1.6) and the NUPOS [[3]](#footnote-3)tag-set. NLTK[[4]](#footnote-4) (Python) used when MorphAdorner encounters a word out of its vocabulary. Java and Python dependencies are not exhibited in the delivered SDK (They are only part of the compilation process).

**LICENSE REQUIREMENT:**

*All SDK artifacts are on github.com:*

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

* 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 2021, is provided also at the bottom of this page.

**IMPROVEMENTS & CAVEATS:**

* Fundamental SDK format has stabilized and is substantially similar to the 2017 and 2018 editions.
* Another two fields have been added to AV-Writ which provides a precise Part-of-Speech representation and lemmatization for each word.
* AV-Lemma has also been updated in the 2020 SDK. Moreover, the binary format of AV-Lexicon is also substantially different from earlier editions.
* Part-of-speech (POS) bits were introduced into the SDK with the HA29 release. As of the Z07 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 to some degree).
* The sqlite lexicon has been eliminated from the SDK.

**ADDITIONAL RELEASE NOTES:**

#1 The “Z-series” edition of the SDK introduced an updated revision number from earlier editions. Digital-AV revision numbers now use a three-digit character sequence, plus an optional suffix/subscript. All revision numbers now 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*=1) represents 2021; (*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/subscript may also be used. If the subscript is a Greek letter (α or β), then this identifies an alpha or beta release of the SDK. Otherwise, a suffix/subscript identifies the discrete date of the release, encoded in base-32; the 1st is 1, the 31st is v.

#2 Two revision numbers exist: Digital-AV SDK revision (aka, the “plate” revision) is the most significant set of files. Not all files in this SDK are required to produce working bible software. Incidentally, the sample source code provided in avx.go implements a minimal set of SDK artifacts, while still providing access to the entire AV Bible text. The avx.go sources use a slightly older version of the SDK (Z08 Revision) and would require minor adaption to update to this latest revision.

#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 Z31 SDK release is substantially identical/compatible with the previous Z14 release, the SDK now includes an addendum for Flat Buffers IDL and binary files

#5 The Z31 SDK release adds direct support for Rust, C, and FlatBuffers. Each addendum describes the specialized support and overall status. Code-Generation is utilized to extend support. These addendums are supplied by the dotnet project named SerializedFromSDK.csproj in the Z-Series/FB folder within the github repo.

**LICENSE:**

Copyright (c) 1996-2023, Kevin Wonus

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice, namely AV-License.txt, shall be included with all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Additional information available at: <http://Digital-AV.org> , <http://AVText.org> , [info@avtext.org](mailto:info@avtext.org) , [kevin@wonus.com](mailto:kevin@wonus.com)

If the developer is willing to take on the dependency of FlatBuffers[[5]](#footnote-5), the deserialization can be driven using the IDL provided in the FlatBuffers Schema (fbs) IDL files. All binary-content files for deserialization for FlatBuffers have an “.data” extent. Corresponding IDL files have an “.fbs” extent. The layouts are substantially similar to the baseline SDK. Therefore, the baseline SDK documentation can still be consulted. However, the IDL (i.e. \*.fbs file) can drive deserialization through the FlatBuffers code-generated sources for most programming languages.

The files in the table below are consistent with the latest revision of the baseline SDK. The fundamental difference is the serialization format itself.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Baseline**  **AV SDK item** | **Baseline Size** | **Flatbuffer**  **IDL** | **FlatBuffer**  **binary content** | **FlatBuffer Size** |
| AV-Writ.dx | 17 mb | Written.fbs | Written.data | 18 mb |
| AV-Book.ix | 3 kb | Book-Index.fbs | Book-Index.data | 1 kb |
| AV-Chapter.ix | 12 kb | Chapter-Index.fbs | Chapter-Index.data | 1 kb |
| AV-Verse.ix | 122 kb | Verse-Index.fbs | Verse-Index.data | 122 kb |
| AV-Lemma.dxi | 179 kb | Lemmata.fbs | Lemmata.data | 742 kb |
| AV-Lemma-OOV.dxi | 8 kb | Lemmata-OOV.fbs | Lemmata-OOV.data | 44 kb |
| AV-Lexicon.dxi | 241 kb | Lexicon.fbs | Lexicon.data | 1.2 mb |
| ­AV-WordClass.dxi | 1 kb | WordClasses.fbs | WordClasses.data | 3 kb |
| AV-Names.dxi | 60 kb | Names.fbs | Names.data | 292 kb |

The FlatBuffers-special files identified above can be found in the FB sub-folder of the Z-Series SDK[[6]](#footnote-6). These files have been written using FlatSharp[[7]](#footnote-7). As of the date of this documentation, all FlatBuffer assets should be considered Alpha-quality. They are available for use, but completely untested as yet.

The status of FB support is entirely experimental and currently moth-balled until interest develops.

In general, the Rust generated files in Digital-AV/z-series/foundations/rust/ folder on github, define structures in lockstep with the binary files of the SDK. However, one major deviation is that the AV-Writ.dx file is segmented into 66 different structures (one for each book of the bible).

There are other minor deviations that should be intuitive by examining the struct definitions. These are driven somewhat by the rules of Rust. The value of the generated files is that no deserialization operations are required. Moreover, the implementation uses Rust arrays with static initializations.

These sources are in flux and undergoing active development. If you encounter problems utilizing these sources, please let me know.

In general, the C/C++ generated files in Digital-AV/z-series/foundations/cpp/ folder on github, define structures in lockstep with the binary files of the SDK. There are some minor deviations that should be intuitive by examining the struct definitions. The value of the generated files is that no deserialization operations are required. The SDK is manifest using static C++ initializers.

The C++ generated sources are ahead of the Rust sources with respect to compilability. If you encounter problems, I would be excited to directly engage with you.

1. OOV stands for “Out of Vocabulary”: Not all lemmas are in the AV-Lexicon; these OOV words can be looked up in the AV-Lemma-OOV table. As an example, “covenantbreakers” is in the KJV bible and therefore in the lexicon. However, covenantbreaker is not in the KJV bible (It is an example of an OOV word). [↑](#footnote-ref-1)
2. <http://morphadorner.northwestern.edu/morphadorner/> [↑](#footnote-ref-2)
3. <https://github.com/kwonus/Digital-AV/blob/master/z-series/Part-of-Speech-for-Digital-AV.pdf> [↑](#footnote-ref-3)
4. <http://www.nltk.org> [↑](#footnote-ref-4)
5. See <https://google.github.io/flatbuffers/> [↑](#footnote-ref-5)
6. See <https://github.com/kwonus/Digital-AV/tree/master/z-series/FB> [↑](#footnote-ref-6)
7. See <https://github.com/jamescourtney/FlatSharp> [↑](#footnote-ref-7)