# Chapter 4: Proliferation of Interfaces

To differing degrees and through different materialities, we have seen the affordances of high surface area, collaboration, and anarchy at work in bible as interface in books and in an early attempt to extend beyond the book in the web Sinaiticus interface. In this chapter, I look toward bible interfaces that move decidedly beyond book, meaning these interfaces are no longer governed by the binding together of pages in a singular volume. As we have seen in other technological shifts in interface, such as the move from roll to codex, these emerging bible interfaces do not entirely erase or leave behind the structures of the page. The interfaces discussed in this chapter deploy the advantages afforded by digital, web-based technologies to a much greater degree than the digital interfaces we have seen thus far. As a result, there is potential for users to engage with the affordances of high surface area, collaboration, and anarchy in ways that exceed the book interfaces of the past.

Extending the analysis of bible interface into the realm of emerging digital technologies[[1]](#footnote-1) involves two important shifts that will demand some new imagination on our part. First, as bible distances itself from the materiality of the page, it is important to understand the continued materialities of emerging digital technologies. Approaching these digital technologies as interface and attending to their affordances reinforces the materiality of the digital, since affordances describe the possible relationships between a user and the material characteristics of an environment or platform. In no way are we leaving the material realm of books and entering some immaterial virtual world of computers and the internet.

As I mentioned in the introduction, digital technologies have multiple layers of materiality. Computers and mobile devices are made of material components like capacitors, wires, precious metals, etc., and the internet is dependent on vast amounts of electricity to power connected devices but also to power the cooling systems that keep the equipment in good shape. While these materialities are necessary to keep our machines running, this is not the materiality I am interested in here.

I am interested in the layers of structure that produce digital technologies and that inevitably impact both the possible and the more probable relationships with a user. This is a digital materiality. In the previous chapter, the digital materiality of the web Sinaiticus interface was visible to the user through the four structured sections of the web page and the limited navigation options. On a different level, typically not visible to most users, the HTML of the page and the XML of the transcription also play a role in the materiality of the interface, structuring the possible uses available in interface.[[2]](#footnote-2)

Awareness of and facility with these often hidden materialities of digital interfaces and their relationship to the other layers of materiality involved in the production and use of digital interfaces is quickly becoming a necessary set of skills demanded of biblical scholars who wish to study bible in the emerging technological landscape. To explore the kinds of capacities that might be needed to work closely with the materialities of bible interfaces at the programming and database level, I will analyze two digital bible interfaces, the Digital Bible Library and the BibleSearch Application Programming Interface (API), the latter of which is built upon the former.

The second shift involved in a move toward exploring digital bible interfaces is a growing attention to the proliferation of interfaces. By proliferation, I simply mean the continual building of interface upon interface, which in turn could give rise to another interface, and so on. Emerging digital technologies highlight the proliferation of interfaces that bible can become. In one sense, proliferation of interfaces is not a new phenomenon for bible. Wachtel’s suggestion that Codex Sinaiticus may have become an exemplar for scribes to use to produce additional copies of the manuscript illustrates one kind of proliferation of interface.[[3]](#footnote-3) Given the broad definition of interface as relationality irreducible to consumption articulated in chapter one, we could also imagine a proliferation of interfaces from the study of a codex bible by a preacher (interface 1) that gives rise to a sermon that preacher gives on a bible passage that is pondered by a parishioner (interface 2), who takes that bible verse and writes it on a protest sign in a march against immigration (interface 3). The sign with the verse gets photographed, tweeted, and discussed on the internet for days (interface 4).

One of the characteristics of the interfaces dealt with in this chapter is that they participate in an ecosystem that is designed to facilitate the ongoing proliferation of bible interfaces as much as to facilitate particular user encounters with bible. This value on enabling the ongoing construction of interfaces also points out that emerging digital bible interfaces may have both human users and machine users at different layers of the interface. For example, the XML of the web Sinaiticus interface was designed for both human users (programmers had to be able to produce, read, and understand the XML) and machine users (their site generator software had to read the XML to produce HTML for the web page). In this chapter we will see an example of a bible interface (BibleSearch API) built upon another bible interface (Digital Bible Library) explicitly to enable the design of additional bible interfaces by users around the world.

## The Bible of Bibles

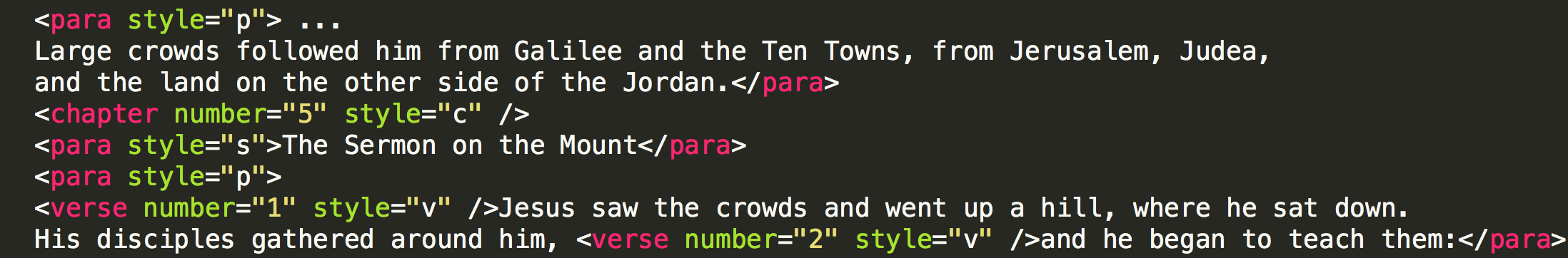
One bible interface that decidedly moves into a digital materiality is a recently emerging project called the Digital Bible Library (DBL). At its simplest, the DBL operates as a giant storage container that can hold any item related to bible.[[4]](#footnote-4) At the time of their public April 2017 Content Update, DBL contained 1508 text entries in 1145 unique languages and 444 audio entries in 374 unique languages.[[5]](#footnote-5) The governing impulse behind this bible repository is to make the content of the bible, “Scripture translation content,” available in “every language spoken on earth.”[[6]](#footnote-6) For all of the language of validation, safeguarding, and authorizing in the DBL mission and vision statement, there is little awareness of the impact of interface on the relationship between users and content.[[7]](#footnote-7)

Recall from chapter one that an interface is a process of interaction among two entities that is irreducible to simple consumption of one entity by another. Bible interfaces in particular involve the interaction of users with technological platforms to participate in the texts and traditions of Judeo-Christian sacred writings. As a technological platform affording participation in several versions of these texts and traditions, DBL fits easily within our concept of bible interface, even if not used in the typical readerly fashions we have seen in previous interfaces. There are two reasons why I have chosen to look closely at DBL as a bible interface. First, DBL provides the backend data interface for one of the most widely used bible interfaces in the world, YouVersion’s Bible App. As a database that provides structured bible translations for the Bible App, DBL is a part of the materiality of this ubiquitous bible interface that goes largely unnoticed or even considered by users or scholars of bible. Second, DBL provides a platform for proliferation, allowing others to create any number of bible interfaces using the artifacts and data available in the library. In fact, DBL itself is a proliferation of interfaces.[[8]](#footnote-8)

While the licensing and distribution aspects of DBL still operate in a paradigm primarily governed by the ideologies of book production, the interfaces enabled by DBL are pushing bible beyond its long history oriented around the page. After an introduction to the technological and cultural situatedness of DBL, I will look closely at another interface that interfaces with DBL, the American Bible Society’s BibleSearch interface. In different ways, these two bible interfaces provide a definitive move beyond the book for bible, while continuing to afford high surface area, collaboration, and anarchy through their use of emerging technologies.

## Bible in XML

On a closer look at the material structure of DBL, it is a collection of XML files that becomes the primary bible interface of DBL.[[9]](#footnote-11) A sample from a DBL bible interface looks like this,



Like we saw with the transcription data of the Codex Sinaiticus web bible interface, this release bundle data in DBL is encoded in XML (Extensible Markup Language). The materiality of this XML code becomes a part of the material limits and possibilities of the bible interfaces built from the DBL. The users of this XML are both human programmers, who need to be able to read this code in order to build interfaces from it, and other machine interfaces that will process this XML code to construct the bible interface used by more typical bible users looking to read the contents.

The syntax of XML resembles that of HyperText Markup Language (HTML), the more familiar standard that web browsers read to present web pages to a user. But there are two important differences: 1) HTML is used to display data, while XML is used to describe data and 2) HTML tags are fixed and standard, while XML tags are flexible and defined by users. HTML is a set of highly standardized encoding tags to tell web browsers how to display and format data on the screen. XML is a flexible set of encoding tags to structure and describe a data set, with little to no regard for how data should be displayed. There are, however, advantages to using a standard encoding practice to allow for predictable processing across multiple documents, and thus many communities of XML users have devised such standards. The XML used by DBL is a specialized encoding standard designed specifically for digitizing biblical texts called Unified Scripture XML (USX).[[10]](#footnote-12) This particular USX standard holds together the versification needs of bible data and continues to contain encoding for structural and style definitions that might be useful for producing print versions of the digitized text. This is another example of how new interfaces reproduce features of their predecessors.[[11]](#footnote-13)

The way the XML of a DBL bible works is through a hierarchical structure of tagged entities with attributes and content. In the example above, the tags are designated by the brackets “<>,” which define “paragraph,” “chapter,” and “verse” tags in this particular selection.[[12]](#footnote-15) As I mentioned above, XML allows for developers to create any tag they need, so the naming and structuring of tags can tell us something about the assumptions and values of an interface. Given the typical difference between XML and HTML, where HTML deals with displaying data and XML with describing data, the presence of paragraph(<para>) tags in this DBL XML is surprising. Chapter and verse tags make sense, since these would be describing and labeling the kind of data in the interface. A paragraph tag has more to do with display than with type of data.[[13]](#footnote-16) Yet, in the DBL bible interface, paragraph operates as a data descriptor at the same level in the hierarchy as book and chapter elements. The prominence of the paragraph element in the XML of the DBL bible interface suggests that though this interface has made a move away from the dominance of the page as the primary unit of encounter for a user, the structural and stylistic demands of organizing text on a page remain a part of the dataset that constitutes bible.

If we look at the style attribute values of the paragraph element in this USX standard used by DBL, we can see the demands of print production of bibles at work.[[14]](#footnote-17) In XML, attributes are additional data that indicate specific details about a given element defined by a tag to offer additional data to describe this element. In the example above, each <para> element has a style attribute indicated by the keyword “style” inside the brackets that define the tag followed by an equals sign and a value, such as <para style=“p”>. A given tag can have multiple attributes that offer different kinds of additional information about the element. For example, the verse and chapter tags in our sample have both a number attribute and a style attribute. The style attribute in the paragraph elements of DBL XML functions much like the style definitions in a word processor like Microsoft Word. In Word, a user can select a style of “Heading 1” or “Quote” or “Footnote” for a particular selection of text in the document and Word will apply a predefined set of formatting rules to that unit of text, such as font size, emphasis, and distance from the margins, to name a few. The predefined set of possible values for the style attribute of the paragraph element in the XML of DBL provide a mechanism of uniform formating for like elements within a bible, much like the heading and quote styles found in Microsoft Word. A <para> style attribute value of “p” indicates a normal paragraph, which signals that no additional formatting is needed for the content of that paragraph element. In our sample XML, we have one paragraph element with style=“s”, indicating a section heading, which can then be treated differently than a normal paragraph when displayed.[[15]](#footnote-18)

The ubiquity of this style attribute in the DBL bible interface indicates a continued investment in the control and standardization of the display of text on a page. The DBL release bundle that comprises the XML contains a styles.xml file that contains explicit definitions for each of the possible values of the style attribute. Users of the interface can elect to adjust these style definitions, choosing a particular font to render normal paragraphs or a different color to emphasize section headings. Yet, the structure of these elements, that is to say, the definition of headings and the beginning and ending of paragraphs, is already cooked into the bible interface through these <para> elements and their style attributes. Due to the involvement of DBL in the production of other bible interfaces in both print and digital forms, the XML encoding of DBL does the work of what is typically divided between XML and HTML, both describing the data of these bibles (XML) and establishing the appropriate parameters for displaying this data (HTML). Including such explicit encoding for display and presentation in the XML of DBL bibles seems to continue to pay homage, perhaps anachronistically or unnecessarily, to the role of the book in defining bible as interface.[[16]](#footnote-19)

## Remnants of the Page

In terms of data description elements, the smallest unit in USX, the DBL XML standard, is the verse. (By elements I mean features that are identified by XML tags.) There are a few smaller elements in the USX hierarchy, such as the character element (<char>), the table cell element (<cell>), and the optional break element (<optbreak>), but as I noted with the paragraph (<para>) tag, these elements are focused on the display of data, not on its description.[[17]](#footnote-20) Since a verse element is always contained within a paragraph element in the USX hierarchy, the words of verses function as the content of a paragraph, with the verse tag simply indicating where a verse number should be inserted in the paragraph element.[[18]](#footnote-21) Though a subtle point, again we see the the page providing the structure for the USX governing architecture; the text is broken up by paragraphs, and the paragraphs contain numbered verses.[[19]](#footnote-22) With the verse as the smallest unit of data description in the XML of DBL and with the content of verses essentially functioning as content of paragraphs due to the hierarchical structure of the elements in USX, the DBL bible interface affords less surface area than it could have if USX had encoded at smaller intervals, such as the word or the character.

We saw a similar reduction in surface area in the XML-encoded transcription of the Codex Sinaiticus web interface. Because Codex Sinaiticus itself was written in *scripta continua,* its fourth century interface required the reader to process the text character by character. The Codex Sinaiticus web interface, on the other hand, encoded the transcription so that the smallest unit was the word, not the character, thereby reducing the surface area. The XML encoding of the DBL has reduced the surface area even further by structuring the text in verse units.

The lower surface area of the XML provides for more efficient processing on the user end. If DBL XML encoded in a higher surface area manner, at the word or character level, then users of the interface (other interfaces such as a mobile bible study application in this case) would have to build in mechanisms for handling these additional surfaces, which could slow down or impede the ability of a user to work through this XML to render the text in an interface. These design decisions toward lower surface area may increase the efficiency of use, but they can also over determine the contours of an interface, which results in less demand on the user to participate in the production of the interface.

## Interfacing Interfaces

DBL is designed with interface proliferation in mind, meaning DBL is an interface for other interfaces. The release bundle XML interface I have just described is intended produce another bible interface such as a print edition, a kindle bible, or a web interface that affords access to several translations at once. Even though the print paradigm encoding governed by the paragraph is present in the DBL interface, a particular user designing a different interface could choose to manipulate the material structures of the XML to resist or alter the conventions embedded in this DBL bible interface. There are explicit suggestions and demands for how a DBL bible is to be used in the construction of a new interface, through the structure of the XML, the style definitions, and the licensing requirements contained in the release bundle. Yet, the attempted persuasive language of “should” used by the documentation for the elements of the DBL XML standard highlights the collaborative and anarchic affordances of this DBL bible interface.[[20]](#footnote-23) Because the XML of DBL does not contain explicit HTML code or complete print layout definitions, the DBL bible interface demands a user to participate in the construction of the bible. This complicated relationship between users and an interface that seeds the development of new interfaces also points to the inability of the DBL XML to control entirely how users will participate in constructing the next layer of interface, an anarchic affordance.

One example of a bible interface built upon and shaped by the digital materiality of the DBL bible interface is the popular mobile bible interface produced by Life.Church called YouVersion.[[21]](#footnote-24) Its utter dominance in the mobile bible market has allowed the YouVersion app to take on the more generic name of simply “the Bible App.”[[22]](#footnote-25) At the time of this writing, the Bible App offers a user just over 1500 bible versions in over 1000 different languages and some of these versions include both audio and text. Shaped by the digital materiality of the Digital Bible *Library*, the Bible App also functions as a library of bibles in a consolidated interface.[[23]](#footnote-26) As we saw above, the XML of the DBL interface has verse as its smallest structural unit. Unsurprisingly then, the smallest unit of text a user can select in the reader area of the Bible App is a verse. A user can select multiple verse objects on the screen, but can not select a single word or phrase within a verse. Here again is an example of the digital materiality of the DBL shaping the material affordances of bible interfaces built upon it.

Leveraging the anarchic affordances of the DBL, the Bible App goes beyond the XML DBL interface and adds high surface area and collaborative affordances through the “Related” feature of the reading area of the app. Once a user has selected a verse, a menu appears with several options for the user to participate in the interface with this particular verse. The “Related” option connects the verse contained in the version file loaded from DBL to other objects in the interface that have been marked as related to the selected verse. Through the Bible App, a user can add a note to a verse and mark this note private, for friends only, or public. If a note is marked public, then any other user can select the same verse and choose the “Related” menu item and see this note. On an iPhone 7, this list of related notes loads in a new screen so the text of the selected verse is not visible anymore. Instead, the user is presented with a reverse chronological--most recent first--list of notes added to this particular verse by users all over the world. This list of notes on a given verse gives a user a point of contact with the larger community of users engaging in the interface, and extends the interface beyond the material limits of the XML of the DBL interface.

As we have seen with a close look at the XML of the DBL, DBL can not be reduced to simply a data or content source for bible interfaces, it is an interface of its own that enacts a digital materiality that shapes the possibilities for interface proliferation. Yet, the vast collaborative capacities afforded by the myriad ways to translate DBL XML into different media interfaces, working with the structural elements of the DBL interface, resists the ability of DBL to dictate or determine use of this bible interface. One of the ways in which DBL attempts to mitigate these anarchic affordances of its bible interface is to closely control who can be a library card holder. The DBL bible interface is decidedly not public. A user, which in this case is an institution such as American Bible Society or YouVersion, has to be a DBL “library card holder” (LCH), which DBL also refers to as a publisher. The LCH must abide by the licensing requirements of a given bible in order to have the ability to download and use a release bundle from DBL. The criteria for becoming a LCH includes

**Reputable Standing** – Only reputable organizations will be considered for participation in the DBL. A reputable organization is one that follows the historic tenets of Christianity

**Cooperation** – While we don’t expect all partners to hold to similar confessional practices or standards, we do expect them all to be able to operate cordially and constructively with all ETEN members, including the inter-confessional ones. Applying organizations should not have any outstanding conflicts with any current ETEN partners or other Library Card Holders.[[24]](#footnote-27)

Unlike the Codex Sinaiticus web bible interface, which was released to any user that has access to the internet, the DBL interface is only available to those approved by the Partnership Credentials Committee. The need for such a process of approval partly has to do with the sensitive copyright information being negotiated in the DBL, but at the same time, it functions as a means of limiting the anarchic affordances built into the interface itself. The four terms listed under the tag line for the DBL on the main page of their website are, in order, “Security,” “Uniformity,” “Availability,” and “Accountability.”[[25]](#footnote-28) The positioning of these terms is important. Availability, the value that connects most closely to the anarchy of the DBL interface, is bracketed by security and accountability. Only in the context of these kinds of controls does the DBL interface champion availability and the close attention to criteria for attaining a library card in DBL supports this kind of minimization of anarchy.

Technologically, DBL affords a reasonably high surface area interface--though lower than some others we have seen--that has expansive affordances for collaboration and anarchy. The collaborative affordances of DBL are most evident in the fact that Library Card Holders (DBL users) themselves determine what is available to other users through the DBL interface. By ingesting their bible translations into DBL and specifying licensing options, DBL users have a high impact on what is possible in the DBL interface. DBL affords anarchy through its capacities for proliferation. Providing a platform upon which users can develop new bible interfaces, the materialities of the DBL interface influence these new interfaces without entirely dictating their limits and possibilities. In practice, there are logistical processes in place that could limit the affordances of collaboration and anarchy in the DBL interface. To see how this negotiation plays out in practice, we will look at one of the early and significant users of the DBL bible interface, American Bible Society’s Bible Search API[[26]](#footnote-29).

## Bible API

American Bible Society (ABS) is a Library Card Holder in the DBL and one of the founding partners. As a Library Card Holder, ABS has built a bible interface called BibleSearch at bibles.org using bibles from DBL to provide a public web search engine for bible and an application programming interface (API).[[27]](#footnote-30) The web search engine of BibleSearch is clean and simple and offers a great deal of functionality for reading and interacting with the textual content of bible.

The more interesting interface offered by BibleSearch is their BibleSearch Application Programming Interface (API), which allows developers to build new bible interfaces using the data BibleSearch has remediated from the DBL interface. Much like a graphical user interface makes it easier for a user to navigate a software application through interaction with graphical representations of objects and functions, rather than resorting to the obtuse syntax of a command line prompt, an API makes it easier for one software application to interface with another. APIs are zones of contact where two software systems interact and negotiate a relationship based on rules and processes defined by the API. Hence the word *interface* in the name. In a growing landscape of application interoperability, APIs are becoming ubiquitous. Google has several APIs, Twitter and Facebook have APIs, Amazon has APIs, and the list goes on.[[28]](#footnote-31) These APIs are what allow programmers to more easily build apps for a mobile device that enables users to check email and post to twitter from their phone. APIs highlight the proliferation possible in emerging digital technologies. With the BibleSearch API, we see a user (American Bible Society) building a bible interface from the bible interface of DBL in order to facilitate the construction of other bible interfaces.[[29]](#footnote-32)

The BibleSearch API offers programmatic access to a select set of DBL bible content for any registered BibleSearch user, which anyone can become. The BibleSearch API is taking secured and authorized DBL bibles and making them available to a much larger public than has access to DBL. Given the copyright and licensing constraints on public access to bibles in DBL, it is no surprise that the number of bible versions available in the BibleSearch API is much smaller than those existing in DBL. The BibleSearch team at ABS is working hard to offer more access in the API, but currently there are 302 versions available in 276 languages, a fifth of the versions made available to the public through the Bible App.[[30]](#footnote-33)

## BibleSearch Interface

So, what does this bible API look like? The best way to see the operations of this interface is by using it, but through the documentation provided by BibleSearch, much can be learned about the API.[[31]](#footnote-34) The introduction to the documentation makes it clear that the API is designed for developers and is not intended for commercial use. The language of the documentation suggests that the API is a means of including “Scripture content and text” in as site or application. Though the data available in the BibleSearch API is oriented around biblical text, the interface can not be reduced to the content available within, it is a framework upon which new bible interfaces can be built. To use this bible API, a user need not meet the complicated criteria for becoming a Library Card Holder in the DBL. All a user needs to do is register an account to identify themselves within BibleSearch and provide a brief description of the interface/application being built to use the BibleSearch API. These barriers to use are far lower than for DBL, allowing exponentially more users to participate in the interface. Like DBL, the BibleSearch API administrators do provide some stated guidelines for use to help mitigate the potential anarchy of the openness of the interface. The second in their list of frequently asked questions, titled “How can I use the data?,” says,

You cannot use the Scripture content retrieved from the API for any commercial purposes, nor in any application that is illegal, offensive, obscene, violent, immoral, or derogatory towards others or the Scriptures itself. Basically, play nice. Don’t alter or change the meaning of the Scripture, or how it could be interpreted.[[32]](#footnote-35)

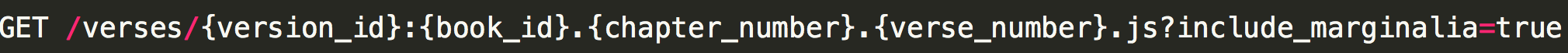
The focus on bible as content here is pronounced, but there is also a distinct awareness of the vast possibilities for use of this bible interface. Again, a caution about use of bible is not a new phenomenon in our emerging technological landscape. It doesn’t take much effort to recall violent and derogatory uses of earlier bible interfaces throughout history, such as war, slavery, and oppression of difference. Yet, that a developer could use this API to build a new bible interface that enacts or propagates these harmful adjectives to an uncontrollably large internet audience does perhaps introduce a heightened scale into the caution here. The final clause of these brief guidelines for use betrays an utter lack of any concept of the importance of interface in the construction and use of bible. Even if bible were reducible to content alone, this articulation of the already established and static meaning and possibilities for interpretation of bible devalues the role of readers as users.

The BibleSearch API is a read-only API that provides either XML or JSON (JavaScript Object Notation) results through HTTPS (Secure HyperText Transfer Protocol) GET requests. Let us look more closely at each of these technical aspects of the API, in reverse order. HTTPS GET requests are what we run when we enter a web address into a browser address bar or click on a link, asking (GET) for a server somewhere to return to us a page and its data in a web browser.[[33]](#footnote-36) Like we saw in the DBL bible interface, the BibleSearch API provides users access to bible via XML or an alternative data encoding language called JSON. Like XML, JSON is self-describing hierarchical data description notation, meaning the elements in a JSON data set and their hierarchy are defined by the data itself, not by some external standard. Instead of the open and close tag notation syntax of XML, JSON is structured by a hierarchy of objects that contain key/value pairs.[[34]](#footnote-37) We will see more specific examples of JSON below when we look more closely at the API in use. Finally, the BibleSearch API is read-only, which indicates that this API allows users only to view data; it does not allow them to modify, add, or delete any data in the interface.[[35]](#footnote-38) The fact that the BibleSearch API uses familiar HTTP methods and common data encoding standards for the results of these methods makes this bible interface more accessible to a larger population of users. Limiting the interface to read-only decreases the collaborative affordances of the interface, since users can not make any marks on the interface itself to participate in constructing the space of the interface. On the other hand, the myriad uses that this interface can afford beyond a simple “reading” of the JSON results does allow users to construct interfaces with greater collaborative affordances.

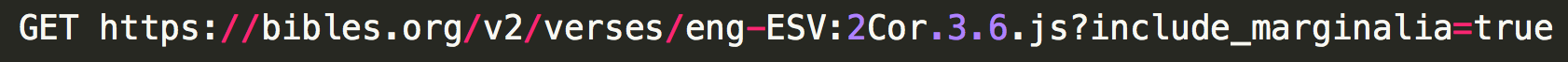
## Requesting Bible

To begin participating in this bible interface, I had to register an account with BibleSearch and then register an application that will be using the API. This registration process provides an authentication token that must be used every time I as a user participate in the interface through the API. Though this registration and authentication provides an efficient and comprehensive means for BibleSearch to monitor a user’s behavior in the interface, the barriers to attaining this access in the first place are very low, particularly compared to the criteria required to become a Library Card Holder in DBL. So, in one sense, the BibleSearch API is providing a public interface for a limited subset of the DBL interface, which is more tightly controlled.

The simplest way to describe the structure of this API interface is that it consists of endpoints that take requests and return responses. An endpoint in this case, such as the verses endpoint, is a URL that provides instructions for a user to request a particular response from the API. For example, to request a specific verse from a particular version with a JSON response I would use syntax modeled by



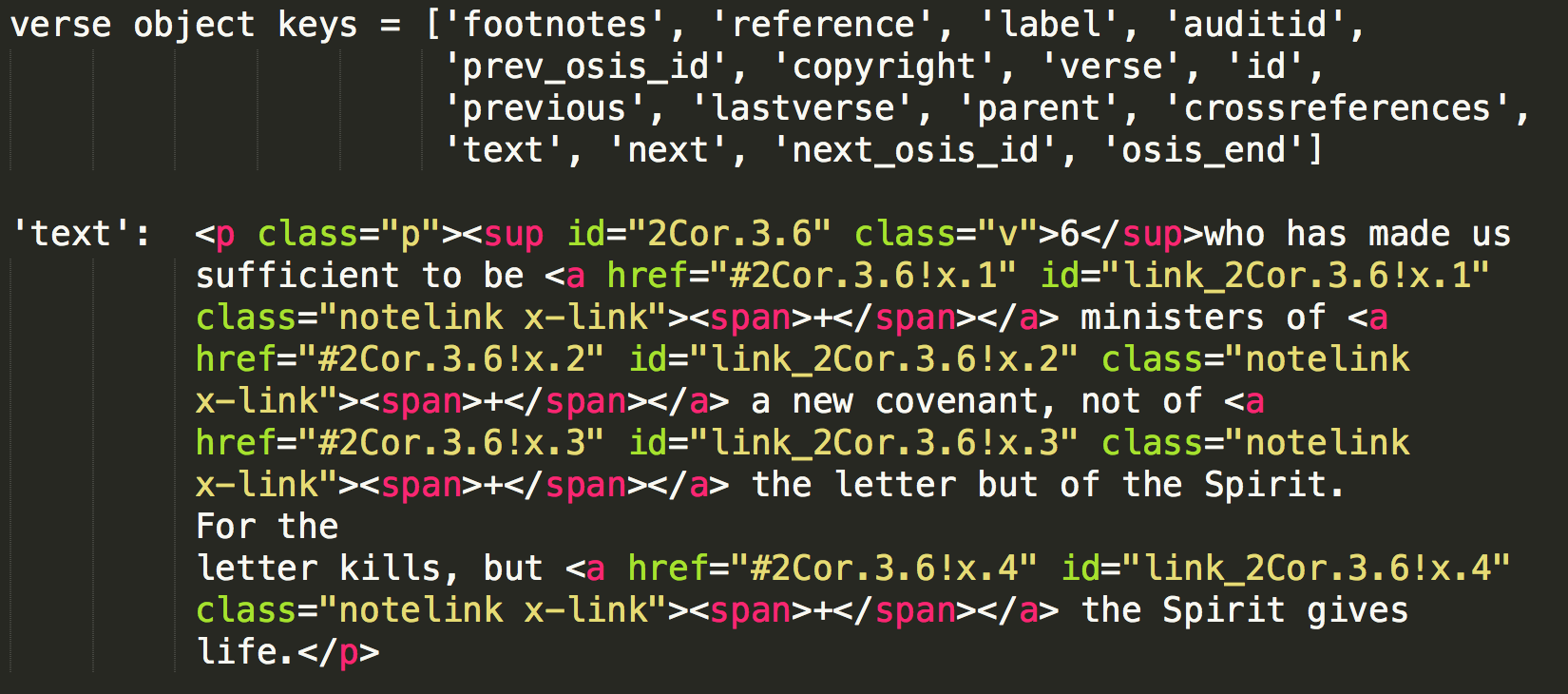
In this schematic, we see the explicit GET method I referenced earlier and then we have “/verses/,” which stipulates the verse endpoint of the API. The rest of the items in this URL are parameters or instructions a user passes to the endpoint to ask for a specific verse. The version\_id is a unique code identifying the specific version from which the verse will be requested.[[36]](#footnote-39) Book\_id is an abbreviation for the biblical book in which the verse is located and these abbreviations are based on the Open Scripture Information Standard (OSIS) definitions for biblical books.[[37]](#footnote-40) Chapter\_number and verse\_number are numerical values that identify the specific verse requested. The “.js” in the URL tells the API that a user wants the results in JSON rather than XML. Finally, “include\_marginalia=true” is an optional flag that can be passed in the API request to include footnotes and cross references in the response along with the indicators for these notes within the text itself. So, if a user[[38]](#footnote-41) wanted to request a JSON response for the sixth verse of the third chapter of 2 Corinthians in the English Standard Version, including footnotes and cross references, the API request would look like:[[39]](#footnote-42)



This API based verse request interface is essentially the same as a verse picker browse function in a web browser, where a user would select a version, book, chapter and verse from a list and select the option to display footnotes. The BibleSearch web interface does just this, taking the user input from the version and chapter selections at the top of the page and constructing an API request much like we have seen above.[[40]](#footnote-43) Here we see BibleSearch itself building layers of interface upon the interface of DBL. The BibleSearch web bible interface is a user of the BiblSearch API, which is a user of DBL bible interface. Bible interfaces in this emerging technological landscape continue to challenge any stark boundaries between user and interface as interfaces become users and users construct new interfaces on the fly.

## Bible as Data Objects

The response from the sample API request we constructed above contains several pieces of data related to the request, including copyright information, next and previous verse identifiers to provide context for this verse and easy machine navigation, the parent object of this verse (2 Corinthians book object), and the verse information including verse number, verse text, and footnotes and cross references. One advantage of an API type interface is that bible is not reduced simply to the textual content of a verse. When a user requests a verse from this API bible interface, such as 2 Cor. 3.6, the interface returns a structured set of data that relates to the verse, such as verse identifiers, the HTML encoded text of the verse, and related footnotes. The verse content itself is only one contact point with this interface, highlighting the digital materiality of this interface and reinforcing the irreducibility of bible to the content of its verses. In the small selection below, I show both the list of keys that make up the JSON response from this verse request and the value of the text key.



The “text” of the verse object is indicated by the ‘text’ key followed by a colon and then the long string following the colon is the text value of the verse object. The ESV text of this verse, 2 Corinthians 3:6 is very short, “who has made us sufficient to be ministers of a new covenant, not of the letter but of the Spirit. For the letter kills, but the Spirit gives life.” All of the encoding in the text of the verse is an HTML translation of the XML provided by the DBL interface for this verse. You will recall the <para> and <verse> elements from the DBL XML had style values of “p” and “v” respectively. In the HTML of the text of this verse object from the API, <p class=“p”> is a translation of the DBL <para style=“p”> ; and <sup id=“2Cor.3.6” class=“v”>6</sup> is a translation of the DBL <verse number=“6” style=“v”/>. The <a> and <span> elements in the HTML construct cross references in the text. The href attribute of the <a> element maps to the “id” attribute of one item in an ordered list of cross reference objects returned by the API in this verse request.

Including cross references like this in the API contributes to the surface area of the interface by providing more points of contact within the verse that interrupt a user’s ability to contain the verse by pointing toward other places in the biblical text. Interestingly, these cross reference objects are not formatted in such a way to make it easy for an application or a user to link from the verse in view to the verses mentioned in the references. Providing linkability by formatting the references using syntax that could easily be passed back to the API would increase the surface area and collaborative affordances of this interface even further by providing avenues for the user to construct different pathways through the text. Cross reference objects are just one example of the many points of contact provided by this API bible interface.

As we saw with DBL, the verse is the smallest unit in the API bible interface, yet the JSON results contain many distinct access points for the verse beyond just the content of the text of the verse, as demonstrated by the keys list in the sample above. The JSON structure of a verse object makes it clear that a verse has many more properties than simply the words it contains. Although each verse object contains data contextualizing it within the larger textual framework, such as next, previous and parent attributes, the sheer fact of constructing verses as objects that a user can engage and use individually foregrounds the typical role of bible as an aggregation of distinct objects rather than a homogeneous whole.

The read-only nature of the BibleSearch API definitely limits the collaborative affordances available in this interface. If a user could write back to the API notes or amendments or other assets that might relate to a verse, such as images, videos, links, etc., this bible interface would provide more opportunities for users to participate in the construction of the space.[[41]](#footnote-44) Yet, the fact that a user has to learn how to construct an API request using some form of application language, such as perl, python, php, etc. demands a level of participation that we may not find in some of the other interfaces considered here. Of course, any bible interface carries with it some kind of linguistic or semiotic demands to engage the text for more than its aesthetic material value. In these emerging digital interfaces of bible that are built primarily as platforms upon which users will build new interfaces, the demands for participation are quite high because a user has to have the literacies and capacities to both interpret the responses from the interface and to construct something meaningful from them. Much like the roll interface shaped some of the possibilities and limits of the codex, so each of these interface layers, such as DBL and BibleSearch API will impact the affordances of the interfaces built upon them. We as scholars will need to continue to develop literacies at each layer of these interfaces in order to critically examine the assumptions at work. It is my hope that this project is an initial experiment in that direction.

## Expanding the Anarchic

As I mentioned earlier, a bible interface that is expressly designed to encourage users to build other interfaces can afford anarchy at a growing scale. Using the BibleSearch API, I could build an interface that constructs an entirely new bible version from a randomized selection of passages from different versions in multiple languages and present this to users in a web browser that allows them to move verses around to construct an entirely different narrative than was presented to them. This kind of use most certainly challenges the reign of an original intention or singular principle governing the interface. This API interface offers great potential for anarchy, even if American Bible Society has offered suggested guidelines for use that are less anarchic and has the ability to terminate a user’s access at any time. Within the structure of the API, there are also some anarchic affordances at work. Several endpoints in the API offer a query parameter or search function that can open up new combinations of verses and interrupt the dominance of the typical order of the text. The API also provides a basic search endpoint that allows for passage or keyword requests. A query based request to the API simply means a user is constructing a question to ask the API, rather than just providing explicit values for parameters to return a specific response. The most interesting of the query based requests is offered in the verses endpoint, where a user can ask any keyword(s) question of the API and get a response that aggregates all verses that meet the criteria provided in the query.[[42]](#footnote-45) For example, if I want to find all mentions of “book” in the bible, I can make a request that looks like[[43]](#footnote-46)



The API response provides a very helpful summary object for this query request that tells me there are eight versions that have 318 occurrences of the keyword “book” across the Old and New testaments.[[44]](#footnote-47) In this single API request, I have constructed a collection of over three hundred verses that cross eight versions and two testaments based on my interests as a user in a single keyword. In four additional lines of code, I can construct a single continuous text of 87,528 characters from a simple concatenation of these 318 verses.[[45]](#footnote-48) Here we see the anarchic affordances of the API bible interface on display as I construct an entirely different text from the results of a keyword search.

## Demanding New Literacies

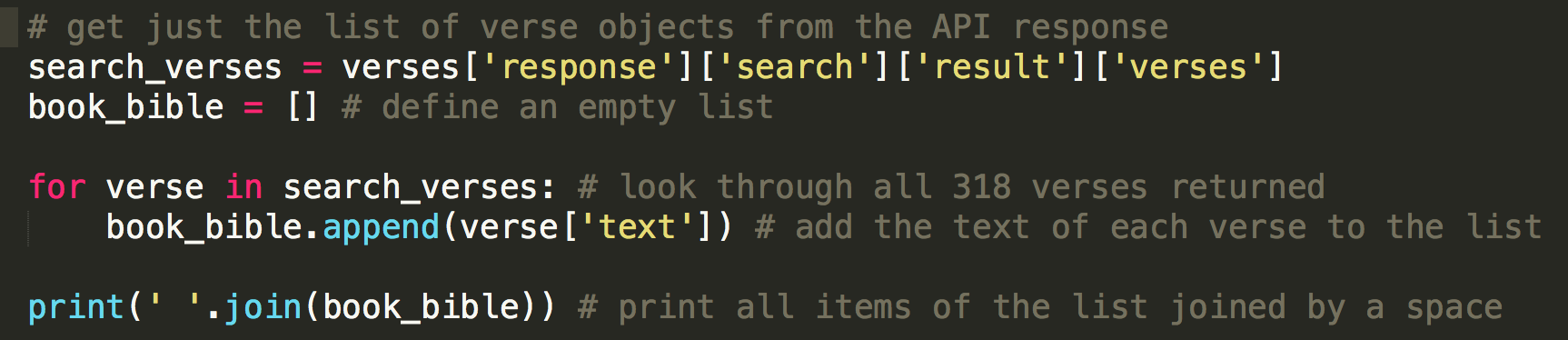
In all of the layers of interface from the DBL to the interfaces constructed by users of DBL, such as the BibleSearch API, we find the affordances of high surface area, collaboration, and anarchy. As bible evolves beyond the technological and cultural categories of book, there is undoubtedly a progression toward higher surface area in bible as interface, even though there is a tendency toward the verse as the smallest unit of the bible instead of the word or character as we found in some earlier book interfaces. The collaborative affordances of book bible interfaces were embedded in the ability of a user to mark the surfaces of the interface through a scribal note or a public user highlight. As bible emerges into a technological landscape that pushes beyond book, its collaborative affordances move toward the ability for users to participate in constructing the interface and make more layers of interfaces, which both DBL and the BibleSearch API demonstrate exquisitely.

The technologies of these emerging interface ecosystems will continue to expand the anarchic affordances of bible by allowing users to build new interfaces that challenge the rule of any original text or intention. At the same time, with the increasing abilities to track usage in internet interfaces, the institutions running the platform bible interfaces like DBL and BibleSearch API will have the ability to police and curtail the extent of experimentation and shape the nature of collaboration based on their perceptions of proper use. Just as biblical scholars have invested in the layers of linguistic interface involved in bible, as bible moves beyond book into emerging technological interfaces, biblical scholars will need to develop the skills to analyze and critique the myriad layers of interface at work in each surface of encounter that is bible. Instead of lamenting the difference and discomfort of bible on the screen, let us take up the challenge to learn new languages to carefully read and build new interfaces for bible as the book relinquishes its reign on knowledge and the bible.

1. By digital, I simply mean technologies pertaining to computing, including things like software, hardware, mobile devices and the internet. I use the language of emerging instead of new to remind us of the ongoing palimpsestuous process of material media translation to gives rise to so called “new” technologies without starting from scratch and never fully leaving previous technologies behind. [↑](#footnote-ref-1)
2. For a thorough outline of the scholarship on digital materiality in the last decade, see Sydney J. Shep, “Digital Materiality,” in Susan Schreibman, Raymond George Siemens, and John Unsworth, eds., *A New Companion to Digital Humanities* (Malden, MA, USA: John Wiley & Sons, Ltd, 2016), 386-396. There are four main categories of digital materiality that emerge from the current scholarship. Matthew Kirschenbaum offers the terms forensic and formal materiality, the former addressing the “evidence” of physical traces left by a digital artifact in processes such as storage and the latter describing the material aspects of a digital artifact that allow for its symbolic transmission, such as the HTML code underlying a web page. In his best attempt to summarize the relationship between these two materialities, Kirshenbaum, *Mechanisms*, 15, writes, “Forensic and formal materiality are perhaps better brought to rest on the twin textual and technological bases of inscription (storage) and transmission (or multiplication).” Building on Kirschenbaum’s two pronged materiality and Jean-François Blanchette’s notion of distributed materiality, Johanna Drucker, “Performative Materiality and Theoretical Approaches to Interface,” *Digital Humanities Quarterly* 7, no. 1 (2013), section 8, suggests a performative materiality for digital artifacts. Much like her work with interface, here Drucker is shifting the focus of materiality from physical properties of a particular artifact to the process of producing materialities in the event that is interface. Drucker’s performative materiality is also close to Hayles’s redefinition of an emergent materiality as “the interplay between a text’s physical characteristics and its signifying strategies” in “Print Is Flat, Code Is Deep: The Importance of Media-Specific Analysis,” *Poetics Today* 25, no. 1 (March 20, 2004): 72. Both Drucker and Hayles articulate a concept of materiality that seems rather close to the notion of affordance at work in our analysis here, which further reinforces the usefulness of affordance as a reminder of the materiality of digital interfaces. I have elected not to use any of the technical categories of digital materiality, because I am worried they will distract from the simpler point that digital interfaces have structures that shape possibilities and limits for use, just as pre-digital interfaces did. Attention to these material structures and the affordances they bring in interface remains a critical part of our work as bible scholars. [↑](#footnote-ref-2)
3. Wachtel, “The Corrected,” 99-100. See my earlier discussion of Codex Sinaiticus and Wachtel’s work in chapter 2. [↑](#footnote-ref-3)
4. In its aim to become the definitive, comprehensive, and authoritative source for all things bible, DBL performs the cultural iconicity of bible as defined by Timothy Beal, *Rise and Fall of the Bible*, Kindle location 66-74. Beal, *Rise and Fall of the Bible*, Kindle location 2006, also helpfully reminds us that bible has always been a kind of library, gathering a cacophony of voices and writings together from different times and places, such that DBL could even be viewed as a bible of bibles. [↑](#footnote-ref-4)
5. https://thedigitalbiblelibrary.org/2017/04/01/april-2017-content-update/, accessed on April 25, 2017. The tracking of the number of unique languages involved in this content highlights the mission and vision of DBL and points toward one of the critical founding partners. The mission and vision of DBL deserves full mention here, “The vision for The Digital Bible Library™ is to contain digital Scripture, in every language spoken on earth by the year 2033. Its mission is to gather, validate, and safeguard Scripture translation content in a standardized digital format, and to empower partners to reach people from every tribe, in every nation, with the power of God’s Word in their heart language by providing authorized and licensed access to the library content.” Digital Bible Library, “About,” https://thedigitalbiblelibrary.org/about/ access on April 25, 2017. [↑](#footnote-ref-5)
6. The mention of every language here signals the deep involvement in DBL of the alliance of bible translation agencies known as “Every Tribe Every Nation” (ETEN). DBL is the primary project funded and managed by ETEN and a quick glance at the homepage of their website, http://www.everytribeeverynation.org/ accessed on April 25, 2017, shows how ETEN locates DBL in the trajectory of technological innovation begun by Gutenberg. The fact that DBL is funded and managed primarily by an aggregation of bible translation agencies helps explain why the content of this library is primarily focused on bible translations, which seems to come with a bias toward bible as reducible to linguistic content. [↑](#footnote-ref-6)
7. See the previous note for the mission and vision statement. Here we have an example of promotion of an interface that falls into Drucker’s critique of mechanistic determinism, taking no account of the participation of the user in constructing the interfaces that are a part of DBL. While DBL provides a mechanism for validation and authorization of the linguistic *content* of a bible, it shows very little interest in controlling the interfaces built around this content except through policing which institutions can hold a DBL Library Card. [↑](#footnote-ref-7)
8. DBL consists of a web interface for participants to manage their licenses, an interface with ParaText, the primary bible translation management software used by translators around the world, to assess and ingest translations into DBL, and an application programming interface that allows users to download translations for use in building other interfaces. For a detailed discussion of the DBL interface ecosystem, see DBL Complete Workflow, https://app.thedigitalbiblelibrary.org/static/docs/about.html#complete-workflow, accessed on September 9, 2017. [↑](#footnote-ref-8)
9. For a helpful look at the overall workflow of DBL and where it is located in the process of bible production and dissemination, see the diagram at http://app.thedigitalbiblelibrary.org/static/docs/about.html#complete-workflow, accessed on April 25, 2017. The technical aspects of DBL are managed by the Institute for Computer Assisted Publishing (ICAP). ICAP is a part of the United Bible Societies, an alliance of National Bible Societies all over the world that manage and produce bible translation projects. ICAP was instrumental in developing ParaTEext, http://paratext.org/, access on April 25, 2017, the software used by bible translation projects globally. Since it is so heavily involved in the translation production process already, ParaTExt has naturally become the primary mechanism for getting items into DBL. In this sense, ParaTExt is one of the many interfaces that participate in DBL. Technically, DBL is considered one part of a larger digital scripture development ecosystem that includes ParaTExt and other tools. See https://thedigitalbiblelibrary.org/about/ visited on April 25, 2017, for more details on this ecosystem approach. [↑](#footnote-ref-11)
10. For detailed description and documentation of the USX 2.5 standard, see http://app.thedigitalbiblelibrary.org/static/docs/usx/index.html, accessed on April 25, 2017. [↑](#footnote-ref-12)
11. The Codex Sinaiticus Project transcription XML had to include a great deal of positional data in the encoding to support the linking between the image and the transcription and the special formatting of the transcription in columnar form on the page. [↑](#footnote-ref-13)
12. All tags in XML have some form of opening and closing. The simplest form of this open/close syntax is demonstrated by the <para> tags in this sample. You will notice that each <para> tag in the code is followed somewhere later in the code by a </para> tag, these mark the opening and the closing of a paragraph, respectively, with the forward slash indicating a closure. Everything contained between these tags is a part of that paragraph element, which can include text as well as other tags, specifically tags to indicate verses. This open and close syntax is identical to HTML. [↑](#footnote-ref-15)
13. One of the core tags used in most HTML standards is a paragraph tag, <p> instead of <para>. So, paragraph is most definitely a part of the encoding for data display. [↑](#footnote-ref-16)
14. For a detailed description of all of the available style attributes in the <para> element, see https://app.thedigitalbiblelibrary.org/static/docs/usx/parastyles.html. [↑](#footnote-ref-17)
15. For examples of the use of the section heading style attribute, see https://app.thedigitalbiblelibrary.org/static/docs/usx/parastyles.html#usx-parastyle-s. [↑](#footnote-ref-18)
16. This detailed attention to stylistic and structural elements in the XML of DBL also reinforces that bible as interface continues to exceed simply the content a bible contains. [↑](#footnote-ref-19)
17. The character element allows for stylistic formating at smaller textual intervals than a while paragraph element. The cell element describes the content and attributes of a single cell in a tabular structure. The optbreak element simply indicates the position of an optional line break in section of text. [↑](#footnote-ref-20)
18. Unlike the paragraph element, which includes explicit open and close tags, the verse element marks only the opening of a verse, not its end. Technically, a verse element can be a child of the <para>, <row>, or <cell> elements, but row and cell elements are only used for tabular display of verses. Instead of including a </verse> tag at the end of each verse, the verse element opens and closes in the same tag, such that technically, the words of the verse are not the content of the verse element, the verse element is empty. The way a verse element indicates open and close in a single tag is to include the forward slash character at the end of the open tag, just before the close bracket. So, instead of <verse>…</verse>, we have <verse … />. In the strictest terms, the verse element has no content, since there is no data in between an open and close tag. A verse element simply has attributes such as number, style, and alternative verse numberings. Each verse tag includes at least a style and a number attribute, with “v” for verse as the only acceptable value for the style attribute and number having a value that indicates the sequential number of the verse within the chapter of which it is a part. Since there is only one acceptable value for the style attribute of the verse element, this attribute is entirely redundant and unnecessary, since the tag name already identifies the element uniquely as a verse. The style attribute is also redundant in the chapter element, which has exactly the same structure as the verse element, but indicates the beginning of a chapter, not a verse, and the chapter element is not a child of the paragraph element. [↑](#footnote-ref-21)
19. When processing these XML files with a language such as python, the text of a given verse is more precisely identified as a “tail” of the verse element. In the Element class of the etree module of the xml python library, a tail in XML is content that follows a close tag. Since each verse tag in USX closes in the open tag and is then followed by the words of the verse, these words are part of the tail of the verse element. Because the verse element is a child of the paragraph element, even its tails belong to the paragraph element, which is why I have articulated the words of verses here as content of paragraphs. [↑](#footnote-ref-22)
20. For an example of this persuasive language, see the additional notes alongside the samples provided for the verse element at https://app.thedigitalbiblelibrary.org/static/docs/usx/elements.html#samples. [↑](#footnote-ref-23)
21. Tim Hutchings has done some detailed description and careful sociological analysis of the religious use of YouVersion’s Bible App in “Now the Bible Is an App: Digital Media and Changing Patters of Religious Authority,” in Religion, Media, and Social Change, ed. Kennet Granholm, Marcus Moberg, and Sofia Sjö (New York: Routledge, 2014), 143–61. In his recent volume, *Liquid Scripture: The Bible in a Digital World* (New York: Fortress, 2017), Jeffrey Siker also considers YouVersion’s Bible app as one example of bible on screen. The Bible App has a counter on their site’s home page that tracks the number of unique devices that have installed the “completely free, with no advertising and no in-app purchases” mobile app. Of course, this number is always expanding, but at the time of writing this sentence, the number of devices having downloaded the Bible App was 270,763,095. [↑](#footnote-ref-24)
22. I will refer to YouVersion’s app as the “Bible App” from now on to indicate the dominance this particular bible interface has garnered in the mobile application space. [↑](#footnote-ref-25)
23. These version statistics are presented to the user of the Bible App at the top of the Versions view in the mobile app. These versions come from the DBL interface I discussed above. Comparing the May 2017 statistics from DBL to the number of versions available in the Bible App, we find that DBL holds 1535 text entries in 1154 languages, slightly higher numbers than included in the Bible App. See http://thedigitalbiblelibrary.org/2017/05/01/may-2017-content-update/#more-917 for the report, accessed on May 8, 2017. [↑](#footnote-ref-26)
24. For a full list of the criteria for being an LCH in DBL, see the LCH criteria accordion menu of the Library Card Holder tab on the Get Involved page of the DBL website, http://thedigitalbiblelibrary.org/get-involved/, accessed on May 7, 2017. There are other criteria, but these two illustrate the non technical attempts to control who participates in this interface. The mention of ETEN in the Cooperation criteria refers to the alliance of ministries and bible translation agencies known as Every Tribe Every Nation. This alliance is the primary governing body and funding vehicle for the DBL. For more information on ETEN, see <http://www.everytribeeverynation.org/>, accessed on October 11, 2017. [↑](#footnote-ref-27)
25. https://thedigitalbiblelibrary.org/, accessed on May 7, 2017. [↑](#footnote-ref-28)
26. American Bible Society, “Bible Search API,” http://webtools.bible/api.html, accessed on September 10, 2017. [↑](#footnote-ref-29)
27. BibleSearch also has a widget tool that embeds a search engine in another site and a highlighter tool that will read a site for bible references and link to BibleSearch content from the reference. For mode details on any of these tools, see http://webtools.bible. Incidentally, ABS was also the driver behind developing the .bible registry for domain names ending in .bible instead of .com or .org, etc. [↑](#footnote-ref-30)
28. For a searchable exploration of available APIs, see https://www.programmableweb.com/apis/directory, which cataloged over 17,000 APIs at the time of this writing. [↑](#footnote-ref-31)
29. DBL has an API that we did not discuss earlier, which allows Intellectual Property Contributors and Library Card Holders easy read only access to metadata (attributes describing the data in a translation) and file downloads for DBL content for which they have a license. For details about the available functions in the DBL API, see https://www.thedigitalbiblelibrary.org/dbl/static/docs/api/index.html#, accessed on May 9, 2017. [↑](#footnote-ref-32)
30. For up to date statistics on available versions in the API, see http://bibles.org/versions\_api, accessed on May 9, 2017. [↑](#footnote-ref-33)
31. http://bibles.org/pages/api, accessed on May 9, 2017. [↑](#footnote-ref-34)
32. http://bibles.org/pages/api, accessed on May 9, 2017. [↑](#footnote-ref-35)
33. The “S” in HTTPS states that the communication between the browser and the server is secured through some form of encryption. HTTP, the non-secure form of these GET requests and HTTPS are supported by the BiblesSearch API, but HTTPS is recommended. The BibleSearch web interface uses their own API to deliver content to a browser based on a user request over HTTP. [↑](#footnote-ref-36)
34. For a brief introduction to the syntax and structure of JSON, see http://www.json.org/, accessed on May 10, 2017. [↑](#footnote-ref-37)
35. The read-only nature of the API effectively limits users to the GET HTTP method and excludes other possible methods such as PUT (edit), POST (add), and DELETE. BibleSearch also suggests that this API is a RESTful API, which means that this is a web service that adheres to the architectural constraints of Representational State Transfer (REST). The REST architecture was introduced in 2000 by Roy Fielding in his doctoral dissertation at University of California, Irvine, titled, *Architectural Styles and the Design of Network-based Software Architectures*, available at http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\_arch\_style.htm, accessed on May 10, 2017. [↑](#footnote-ref-38)
36. For samples of version\_ids and more details on the versions endpoint, see http://bibles.org/pages/api/documentation/versions, accessed on May 10, 2017. [↑](#footnote-ref-39)
37. For more information on OSIS, see https://www.crosswire.org/osis/, accessed on May 10, 2017. [↑](#footnote-ref-40)
38. I have two possible users in mind here. First, we could imagine a programmer designing a bible interface using the BibleSearch API as our user. Second, we could imagine the machine or the code running the newly developed interface as the user of the API. In fact, API is common language to describe interfaces between two pieces of software. So, here we see another sense in which these emerging bible interfaces are moving beyond our typical notions of bible as book, by involving users that are machines. [↑](#footnote-ref-41)
39. The https://bibles.org/v2 is the trunk of all API endpoints including an explicit API version number “v2.” If a request leaves out the explicit version, the request will use the default API version, which is currently “v1.” Versions codes in the API contain both a language indicator and the version abbreviation, such as “eng-ESV” for English language - English Standard Version. [↑](#footnote-ref-42)
40. See http://bibles.org to try the web interface for this API. The main difference in the web interface is that it uses primarily the chapter endpoint to return all verses for a selected chapter of a selected version and it formats the results for viewing and interacting with in a browser. [↑](#footnote-ref-43)
41. The BibleSearch web interface does allow users to tag verses with keywords and add notes to verses. Currently, the API allows read access to the tags associated with a verse but not the notes and there is no way via the API to add or modify tags. For more detail on the tags endpoint, see http://bibles.org/pages/api/documentation/tags, accessed on May 11, 2017. [↑](#footnote-ref-44)
42. For a detailed look at all of the possible search criteria that can be passed to this verse search endpoint, see the Searching section of http://bibles.org/pages/api/documentation/verses, accessed on May 11, 2017. [Finished in 0.1s] [↑](#footnote-ref-45)
43. The python code I use to run this API request is included here:

     [↑](#footnote-ref-46)
44. The summary object looks like this:

     [↑](#footnote-ref-47)
45. The additional lines of code with comments are

     [↑](#footnote-ref-48)