

UNIVERSITY OF CALIFORNIA

Santa Barbara

Participatory Design in Digital Language Documentation: A Web Platform Approach

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Linguistics

by

Patrick James Hall

Committee in charge:

Professor Eric Campbell, Chair

Professor Matthew Gordon

Professor Simon Todd

Professor Alexis Palmer, University of Colorado Boulder

March 2022

The dissertation of Patrick James Hall is approved.

Matthew Gordon

Simon Todd

Alexis Palmer, University of Colorado Boulder

Eric Campbell, Chair

March 2022

Acknowledgements

With some astonishment, I now look back on almost thirty years as a student of linguistics. I was hooked from the first day of LING 5 at Berkeley in 1993, which I somehow managed to find despite the fact that it was in Dwinelle Hall. I still feel a debt to the whole Berkeley faculty, including especially the late John Ohala, whose classes introduced me to phonetic and phonological fieldwork.

Between my undergraduate years and re-entering academia at the LSA Institute in 2009, I worked in the technology world, which of course has its echoes in the present work. Among many friends and colleagues from those days, my brother John was and remains my most stalwart friend. I am deeply grateful to him for his kindness and encouragement over many years.

In 2009 I packed my bags and moved back to California with vague dreams of fieldwork and academia. Somehow I squeezed into the indefatigable Pam Munro's LSA 310 class (I was thirteenth student of twelve slots, whoever dropped the class, wherever you are, I thank you). We had the privilege to study the Kashaya Pomo language with the late Anita Silva. It was a lifechanging experience and one which I think of often. I remember Anita's kind heart and remain moved by her dedication to her language and culture. (I also remember how hilarious she was — upon discovering that I didn't know what an oak gall was, she quipped "Don't get out much, do ya, city boy?")

I also thank Andrew Garrett for providing me with a copy of the digital remnants of Robert Oswalt's research on Kashaya and the other Pomoan languages and to hang

around the Survey scanning Oswalt's Kashaya dictionary. The trajectory to the current work began there.

Some of my fondest memories in linguistics are from summer programs at Breath of Life, the LSA Institute, and CoLang. To all the people who organized, taught at, and participated in those institutes — too many to name — I am deeply grateful. There is no better way to spend a summer than nerding out with linguists. Highly recommended.

And then there was UCSB. It's hard to overstate how grateful I am for the opportunity to have been a part of the Linguistics program there. Obviously the location is a part of the charm of the university, but even had the campus been found on a desolate plain strewn with tumbleweeds, a chance to learn from and work with Marianne Mithun, Mary Bucholtz, Wallace Chafe, Pat Clancy, Bernard Comrie, Jack Dubois, Carol Genetti, Matt Gordon, Bob Kennedy, and Sandy Thompson would have been well worth it.

To my advisor Eric Campbell, it's not possible to express enough thanks. Your open-mindedness as to the worth of this non-traditional dissertation is what made it possible. For your endless patience and advocacy, I thank you, my friend.

I would also like to thank my dissertation committee, including Eric, Matt, Simon Todd, and Alexis Palmer of the University of Colorado Boulder, for their patience, good humor, and insightful advice.

My sincere thanks to Allison Adelman for her guidance, patience, and good humor.

To the true Santa Barbaran, and true scholar, Brendan B. Barnwell, I offer the highest thanks in the form of his own motto: “Hmm, interesting.”

I thank especially Carmen Hernández, Jorge Toledano Ortiz, Arcenio J. López, and the many other members of MICOP and the Mixteco community in California, not only for allowing me to get to know the Mixteco community, but also for showing me what community means.

I want to express my thanks to the memory of Joshua De Leon, Sr., who shared his Hiligaynon language with the fieldmethods class at UCSB, and who was a tremendous friend.

To the many wonderful friends on the Docling Forum, I thank you for an oasis of linguistics and kindness during these dark times.

I am ever-grateful to my parents, who worked so hard to give me the chance to follow my dream, and to my sister, who put up with me!

And last but not least, I wasn’t kidding when I said linguists are fun to hang out with: I married one. Lynnette, you are still my north star. Thank you for everything.

And Sophie, guess what? Daddy finished his book!

VITA OF PATRICK JAMES HALL

March 2022

EDUCATION

Bachelor of Arts in Linguistics, University of California, Berkeley, May 1997

Master of Arts in Linguistics, University of California, Santa Barbara, June 2014

Doctor of Philosophy in Linguistics, University of California, Santa Barbara, March 2022 (expected)

PROFESSIONAL EMPLOYMENT

Teaching Assistant, Department of Linguistics, Winter 2013

Teaching Assistant, Department of Linguistics, Spring 2013

Teaching Assistant, Department of Linguistics, Fall 2014

Teaching Assistant, Department of Linguistics, Winter 2015

FIELDS OF STUDY

Major Field: Language Documentation

ABSTRACT

Participatory Design in Digital Language Documentation: A Web Platform Approach

by

Patrick James Hall

Software for language documentation has a long and successful history, but challenges remain. In this work I describe my efforts to provide a path toward reinvigoration of software design for language documentation, one which equips linguists to participate more directly in designing software for their own workflows. I begin with the description of a simple, extensible data model for all the major data types used in language documentation. I then show how the standardized, well-tested technology that powers modern web browsers can serve as the basis for novel interfaces which are nonetheless closely linked to the terminology and descriptive systems in which linguists are trained.

Contents

1. Introduction.....	1
1.1. ‘Large, windless apartments’: History, technique, and technology.....	8
1.2. Dataflows, Workflows, and Components	15
1.2.1. Some sample applications	16
1.2.2. ELAN and time-alignment.....	20
1.2.3. An alternative approach to time-aligned lexical data: an iterative recording workflow.....	23
1.2.3.1. Iterative recording interface	27
1.2.4. From documents to applications: A functional overview of HTML and the web platform	28
1.3. Custom elements for language documentation: <code>docling.js</code>	37
1.4. On motivations and voice	39
1.5. Outline in brief.....	43
2. Data types and dataflows: Building a Boasian Database.....	45
2.1. Toward a Boasian database	45
2.1.1. Designing a documentary database.....	45
2.1.2. A shift in viewpoint: from streams of text to databases of objects	49
2.1.3. Beyond hyperlinks	53
2.1.4. What is database design?.....	57
2.2. An object model of documentary data, and a JSON implementation	62
2.2.1. Precedents	62
2.2.2. Machine-readable data types with JSON: attributes, objects, and arrays.....	63
2.2.3. Documentary data types	69
2.3. Designing a Boasian Database.....	73
2.3.1. A catalog of data types in <code>docling</code>	74
2.3.1.1. Language Data Type.....	74
2.3.1.2. Word Data Type	76
2.3.1.3. Sentence Data Type.....	79
2.3.1.4. Text Data Type	81
2.3.1.5. Corpus Data Type.....	82
2.3.1.6. Lexicon Data Type.....	83
2.3.1.7. Grammar Data Type.....	91
2.4. Modeling data change with dataflows	92
2.4.1. Familiar documentary dataflows.....	93
2.4.1.1. Elicitation	93

2.4.1.2.	Glossing	94
2.4.1.3.	Time-alignment.....	95
2.4.1.4.	Word Exemplification	96
3.	Viewing Documentary Data	100
3.1.	Reviewing key concepts	106
3.2.	A first look at web components	107
3.2.1.	A typology of web components for docling.js.....	113
3.2.1.1.	Views	114
3.2.1.2.	Lists	115
3.2.1.3.	Editors	116
3.3.	Displaying data with view and list components.....	117
3.3.1.	Granularity in digital documentation.....	120
3.3.2.	<word-view>	122
3.3.3.	<sentence-view>.....	125
3.3.3.1.	Granularity and responsiveness in the <sentence-view> component.....	128
3.3.3.2.	The structure of <sentence-view> markup.....	131
3.3.3.3.	Accessibility is a universal concern	133
3.3.4.	<word-list>	134
3.3.5.	<metadata-view>.....	137
3.3.6.	<lexicon-view>.....	140
3.3.6.1.	Sorting	140
3.3.6.2.	Search.....	140
3.3.7.	<sentence-list>.....	142
3.3.8.	<text-view>	143
4.	Components for documentation workflows.....	145
4.1.	Designing user interfaces for documentation workflows	145
4.1.1.	Static documents vs. interactive interfaces	151
4.2.	Transcription	154
4.2.1.	Manuscript transcription.....	158
4.2.2.	Finding characters and phones	162
4.2.3.	A user interface design for inputting phonetic transcriptions.....	164
4.2.4.	Inferring phonetic inventories from transcriptions.....	167
4.3.	From workflow to interface	168
4.3.1.	Scheduled vocabulary	168
4.3.1.1.	An aside on representing grammatical categories	173
4.3.2.	Image prompts: sharing an interface between both participants.....	178
4.4.	Harmonizing time-aligned data	180
4.5.	Automatic data updates depend on User Interface Design	186

5. Avenues ahead	189
5.1. The Web Platform and <code>docling.js</code>	190
5.2. Fostering a documentation interface design community	195
5.2.1. Designers.....	196
5.2.2. Implementers.....	199
5.2.3. Users.....	201
5.2.4. Documentation histories: Certainty and revision	202
5.3. An open-access online course and next steps	204
5.4. Conclusion	206
6. References.....	208

1. Introduction

“In the beginner’s mind there are many possibilities, but in the expert’s there are few.” ([Suzuki 2010](#))

The field of language documentation has relied for decades on a small number of software applications. While these tools are excellent in many regards, their usage has been traced out into trails which have since been paved into hardened expectations about how linguists should interact with digital information. Here, we will attempt to look at the task of managing digital documentation with fresh eyes, by approaching the design of software for language documentation directly.

This work provides answers to questions such as these:

- **What is documentary data?** What is its shape? How can I store complicated nested information in a useful way?
- **How can I get documentation onto the web?** How can I use the internet to display data in a familiar way?
- **I have an idea for a better user interface, what should I do next?.** My work would be much more efficient if could just work with my data in a particular way. What can I do to get my idea implemented?

Answering such questions is not simple because, after all, software is complicated. As we shall see below, the web platform we will be building in has many moving parts. But gaining a systemic understanding of how the web platform works as a whole is empowering. Furthermore, we don’t have to start from scratch. We will

be discussing the use of a software library (`docling.js`) which is designed with many kinds of users in mind.

Chapter 5 introduces the notion of *participatory design*, but here are some guidelines as to how three kinds of readers might make use of this work.

- **Users** If you are a working documentary linguist who makes use of software in your work, but who doesn't have time to take on a more in-depth plunge into the technical details of software, this work can nevertheless serve as a useful bird's-eye view of how web technology can help make software more useful for you in the future.
- **Designers** If you are one of the many documentary linguists who the degree to which the current generation of documentary software allows you to flexibly handle your data and workflows, then this work is for you. My goal is to equip readers like you with the information you need to start participating in the process of designing software to meet your own needs, in collaboration with your colleagues.
- **Programmers** (or “implementers”, see §5.2.2) If you are already a programmer, `docling.js` offers a working prototype of one approach to implementing user interfaces for documentary data and workflows. You can examine the source code at our Github Organization: <https://github.com/orgs/docling-js>.

It is also designed to have more than “level” of learnability. Learning how some simple “custom HTML tags” such as `<text-view>` can be inserted into an HTML file in order to display an interlinear text is a much easier task than learning how that

custom element actually works behind the scenes. One might not need to know how a transmission works in order to drive a car, but knowing the difference between “drive” and “reverse” is useful!

I have striven to make the technical topics described here — and there is no denying that software design is a complex topic — as understandable and relatable as possible. By reading this work, the reader will not become a programmer. They will, however, take what amounts to the first step in becoming a programmer: learning to describe exactly what software should do. It is hoped that the kind of work we will engage in here will strike the reader as “the good stuff” as far as linguists are concerned — all examples here (apart from a brief foray into vintage cars in Chapter 3!) are linguistic in nature, and all are ultimately concerned with building interfaces that will seem both familiar and useful to working documentary linguists. Like language documentation, software design is a craft best learned in the company of others. More than anything, I hope that this work provides an entry point for those who might feel overwhelmed by the current technical landscape in documentation, but who also feel that “soldiering on” with existing tools is the only way forward.

So what is the nature of the problem which this work seeks to address? Ultimately, I see the gradual, ongoing process of digitization of language documentation as one which should strive to incorporate the long history of expertise which has been developed by the discipline. Language documentation already has a well-developed “mental model” of the nature of its data, and a broad array of methodological tools for collecting, managing, and re-using such data. However, our current generation of tools were not designed to work together — this

makes unifying and handling data created with distinct tools a chore. We must meet this need by creating a coherent, complete, and extensible data model which covers all of the types of data we work with in documentation, but this is only the aspect of the work described here. We must also take into account the ways in which language documentation is fundamentally a collaborative activity: we learn to *do* documentation by working *with* other linguists. Software designed to expedite language documentation, therefore, must be designed with those collaborative working methods in mind, as articulated by experienced language workers.

There is a sizeable didactic literature on the methodology of language documentation: just taking a few references from the domain of fieldwork manuals, notable works include Samarin ([1967](#)), Hale ([2001](#)), Crowley ([2007](#)), Chelliah and De Reuse ([2010](#)), Sakel and Everett ([2012](#)), and Bowerman ([2015](#)). However, there is an important contrast between the kinds of information that such works contain and the current discussion within the discipline about how digital materials should be managed. Most of our field's institutional focus on digital materials is now on how to handle completed language documentation through archiving. Despite significant current interest and work on archiving language documentation, there is less robust discussion of integrating the design of software for doing language documentation. Put simply, as far as discussions of software go, the field of language documentation has been more concerned with documentation than documenting.

Documentary data is in fact less ephemeral than the software used to create and manipulate it. While we will be relying here on the web platform as a basis for designing, using and distributing software and digital documentation, it is

nonetheless true that the web platform itself is subject to (and indeed, expected to) change over time. We should think of the true currency of documentary work as data files themselves, not software, and not user interfaces — what is canonical is the data, not the user interface. Learning to use software is actually a very commonplace activity — the best software is less noticeable precisely because it is designed in a form that is familiar to the user, and is therefore more intuitive to use. Thus, if we are careful to keep standard linguistic notations and practices in mind as we design and implement software, we can expect that the learning curve for users will be shallow, and thus maintainable. Rather than a single kitchen-sink application that does everything, we can build in flexibility and learnability by using a modular approach where new features are learned independently as they are added to the system.

This is not to critique the focus on archival processes: it is at the point of deposit, after all, that many of the most vexing and challenging issues in language documentation are inevitably foregrounded. How should documentation be distributed? Who determines what the permissions regime for viewing, using, and modifying that documentation will be? What institutions will maintain and fund the archive? These ethical aspects of the creation of language documentation archives are so nuanced and fraught that they have engendered their own ongoing subfield, and rightfully so. Woodbury ([2014](#)) gives guidance on producing “documentations that people can read, use, understand and admire.”¹ Again, this advice primarily

1. See ([Henke and Berez-Kroecker 2016](#)) for this and other sources in an annotated bibliography of archiving in language documentation.

addresses the *use* of language documentation more than its creation — bearing in mind that use and creation of documentary data will always be an iterative process in which one shapes the other. But while such criteria are useful for evaluating completed fieldwork data at the time of deposit, evaluating the practical utility of user interfaces for creating, modifying, and using documentary data in the first instance requires quite different criteria.

It is the linguist's relationship to technology during the ongoing process of collecting and analyzing initial fieldwork data that is the primary focus of the current work. How do linguists actually interact with software *as* they create and modify their linguistic databases of textual, lexical, and grammatical material? How do those technologically-mediated interactions with language compare to the practical advice for how to collect linguistic evidence contained in the large literature on fieldwork methodologies? In other words, how does a documentary linguist's training in field methods intersect with the digital data that they produce and manage as they are working with speakers and analyzing the resulting data?

As mentioned above, it is certainly not the case that there is no discussion of software in the current documentation literature; to the contrary, there is no shortage of software reviews, software troubleshooting advice, workarounds for interoperability between distinct software tools, and detailed accountings of how a particular documentary project was carried out via particular methodologies of processing and re-processing data. What is less commonly discussed are the more general questions of just what functionalities a software ecosystem for language documentation should support in the first place: we seem to have come to a juncture

where a strong association between particular software tools and particular documentary tasks is accepted as default best practice. For instance, the notions that “the tool for time aligned data is ELAN” or that “morphological analysis can be done with either FLeX or Toolbox” are so ingrained in our working culture as to seem practically inviolate. But this quasi-standardization as come to impose a tremendous inertia when it comes to creativity in designing software for doing language documentation.

In this work, I propose a particular approach which I hope will enable more innovation in this space. I intend to demonstrate that user interfaces built with the *web platform* can support the whole process of gathering, analyzing, and reusing all the kinds of data commonly collected in language documentation. The web platform has evolved into something much more powerful than simply a means of interlinking static documents: it is now a widely supported, flexible, powerful platform for not only sharing documents, but also for implementing user interfaces for using all of the information contained in such documents, or as ([Munro 2001, 142](#)) puts it, “manipulat[ing] and arrang[ing]” documentary data. By building directly on the capabilities of the web platform, we can design and implement software tools for language documentation in a modular fashion, breaking down complex documentation tasks into smaller steps.

From there, we can design simple, focused user interfaces which are scoped to specific, familiar steps in documentary workflows as they are carried out by linguists in practice. These smaller, modular user interfaces may then be composed into larger applications that are built up gradually to complete the complex tasks. Our goals will

include addressing related questions: How can we use the web platform to design user interfaces which are custom-designed for particular tasks in language documentation work as it is practiced? How can we marshal the expertise of working linguist to achieve that goal in a cooperative manner? We shall see some preliminary examples of how to address these problems below, but in general terms, we will strive to create an approach to designing interfaces which is *participatory* by prioritizing how documentary linguists *collect* and *use* their data, but also how they *think about* that data and those workflows.

But first, let us consider the “work history” within which the recognition of “language documentation”, and especially digital language documentation, has come to be seen as a distinct field in its own right. How did earlier practitioners of language documentation relate to their data, and how do modern attitudes toward data within the field compare?

1.1 ‘Large, windless apartments’: History, technique, and technology

Swimming as we all are in a sea of information, it is difficult for modern linguists to extract themselves from the present and imagine a time when their language work was not mediated in some way by computers. But a pair of quotes from early practitioners of language documentation bring the sea change from the age of paper to the digital era crashing home. J.P. Harrington’s vast and ramshackle documentary work on the languages of the Americas needs no introduction, but he seems to have published little in the way of direct didactic guidance with regard to his own

methodologies for carrying out linguistic fieldwork and analysis. However, Klar ([2002](#)) recovered an outline by Harrington for a 1914 lecture on fieldwork methodology, perhaps the only explicit record of Harrington's working methods in his own words. In the following quote, he describes how would-be field linguists should sort their file slips:

The sorting of slips should be done in a large, windless appartments [sic] where the work will be undisturbed. Many Large [sic] tables are most convenient, although it often becomes necessary to resort also to placing slips on the floor. The sorting of many thousands of slips is a most tedious and laborious task. To many it is mere drudgery. It cannot well be done by anyone other than the collector. It is easy and interesting to record words on these slips by the thousand. The sorting however takes many times as long as the recording.

It is evocative to imagine Harrington in a bleak rented room somewhere in California, surrounded by his innumerable slips of paper, deathly afraid of a breeze blowing away a year's worth of careful organization of precious notes on Chumash or Ohlone. And he is not the only figure in the history of language documentation to have suffered from this curse of sorting: it also plagued Leonard Bloomfield, whose complaints of the ailment were recounted by Hockett in the introduction to Bloomfield (1967) (emphasis added):

Bloomfield was speaking of the tremendous difficulty of obtaining a really adequate account of any language, and suggested, half humorously, that linguists dedicated to this task should not get married, nor teach: instead

they should take a vow of celibacy, **spend as long a summer as feasible each year in the field, and spend the winter collating and filing the material.** With this degree of intensiveness, Bloomfield suggested, a linguist could perhaps produce good accounts of three languages in his lifetime.

Even Bloomfield himself did not fulfill his suggested agenda: he did not live to see the publication of his one full-scale grammar, *The Menomini Language* — it was Hockett who would edit and see through the posthumous publication of that work (Bloomfield 1969). While there is a tongue-in-cheek quality to Bloomfield's advice, Harrington's true evaluation of his own claim can of course be measured in (mostly unpublished) tons. But however we interpret the detail of these admonitions, their import is clear enough: two experienced documentarians agreed in that aspiring linguists should expect to spend a significant portion of their careers — perhaps one *half* of their careers — engaged in sorting data.

It is humbling to imagine what such productive and prolific linguists could have achieved had they not been constrained in this way. One is reminded of the 18th-century lexicographer Samuel Johnson's self definition as 'a harmless drudge'. Surely our easy modern access to relatively cheap and powerful computers obligate us to view hand-sorting in the era of computers to be drudgery of a "harmful" sort? And yet, some modern linguists seem to exhibit either reluctance to let go of the old paper-based ways of documentation or a lingering unhappiness with the idea that those methods should ever be considered obsolete:

The first dictionary I did (a preliminary version of my Mojave dictionary)

was compiled in three-inch by five-inch slips (some linguists, I know, prefer four-inch by six-inch slips!) - not cards (too thick!), but slips of ordinary paper, which were arranged alphabetically in a file box (one hundred slips take up only a little more than half an inch). Reluctantly, I have stopped introducing field methods classes to the joys of using file slips, which I still feel are unparalleled for their ability to be freely manipulated and arranged in different ways. But I don't use paper slips much myself any more, so it doesn't seem right to require students to make a slip file, as I once did.

([Munro 2001](#))

While Munro's observations should not be taken to imply a lack of enthusiasm for digital approaches to documentation, there seems to be a touch of frustration with the capabilities of digital approaches, precisely because they fail to provide the same kinds of spontaneous sortability that the physical fileslips did. It is worth bearing in mind that the physical artifacts of traditional documentation are themselves a form of information technology, which evolved through experimentation and sharing of expertise. Insofar as our current software interfaces fail to emulate what were once readily available means of "manipulat[ing] and arrang[ing]" information, we should see that as a shortcoming of our *software* which is too simple. In addition to developing innovative kinds of digital representations, we should also seek continuity with the kinds of expertise that developed before computers were readily available.

But even so, would not Harrington or Bloomfield have given a king's ransom for one of today's mid-range, off-the-shelf laptops? Those seeking support for carrying

out fieldwork today, of course, must not vacillate as to whether to work digitally. Funding sources stipulate that corpora be deposited in digital form — fileslips and notebooks alone do not constitute sufficient archival deposits today. As the field of documentary linguistics has quickly expanded, the seven principles for portable documentation as set forth in Bird and Simons’ seminal ([2003](#)) paper (which argues that documentation should be evaluated in terms of *content*, *format*, *discovery*, *access*, *citation*, *preservation*, and *rights*) have grown ever more relevant. But despite this intense activity in ensuring the longevity and usability of the outcomes of language documentation, the feelings of “impotence” and “frustration” seem to persist with regard to *doing* documentation. Many questions arise when we view the theorization of digital approaches to documentation in light of this distinction. Why do documentary linguists so often find that using computers to improve their research is more vexing than helpful? Why is so much of our effort dedicated to troubleshooting software *for* documentation, as opposed to simply *doing* documentation? If we are attempting to do “computer-aided fieldwork”, then where is the “aid”? What is it about the software interfaces we are using in documentary linguistics that results in work that is “joyless”?

I suggest that answers to these questions lie in a kind of skewing between the way linguists think about their data, and the design of the software applications that linguists carry out in order to acquire that data during and after fieldwork. To be helpful to the process of doing fieldwork, software interfaces must be designed not only with specific kinds of data as input and output, but also with a clear definition of exactly what steps will be taken to transform the input into the output. Paraphrasing Munro, how can we create software interfaces for language documentation which

give the same freedom to *manipulate and arrange* data as the old paper-based methods (while still, of course, benefitting from the speed of computers to which Harrington's and Bloomfield's generations did not have access)? As we shall see in the next chapter, the number of kinds of information required to digitize documentary data are less extensive than one might at first expect. They are also reasonably simple, and more or less conventionalized, if not standardized. Linguists understand the structure of their data already. The problem at hand is not to reconceptualize linguists' mental models of their data; it is to reconceptualize the way that those models are encoded into software, and to reconceptualize the approach to creating software for manipulating those models interactively.

Raskin ([1994](#)) makes a relevant observation about what makes a user interface “intuitive”:

...a user interface feature is “intuitive” insofar as it resembles or is identical to something the user has already learned. In short, “intuitive” in this context is an almost exact synonym of “familiar.”

In the context of language documentation, it is problematic that the available software tools for doing language documentation are not intuitive enough: working with most modern software tools for language documentation means bouncing back and forth between multiple applications (many of which are well-designed in their own right) in order to address different aspects of the data model, and different steps in a documentary linguist's workflow. For instance, ELAN focuses almost entirely on the task of timed transcription alignment, and has very limited capabilities as far as a built-in notion of “words” is concerned. Conversely, Fieldworks (also known as Flex)

is in a sense the inverse of ELAN: it has elaborate capabilities for managing the morphological analysis of a corpus, but no ability to sync media recordings with that corpus. This division of labor between software tools (and there are usually other applications involved as well) is “unnatural” to linguists in the sense that they do not think of their time-aligned data as being cloven in two in this way. Time-aligned, interlinearized texts are the basic data type for many kinds of linguistic research — not a *combination* of morphologically analyzed data and time-aligned data. Similar mismatches could be enumerated with regard to many of the other software applications which serve particular niches in documentation: Praat has robust capabilities for phonetic analysis, but its use is not integrated into the workflows for FLEx or ELAN.

And it is perhaps this kind of mismatch between software and mental models that underlies the frustrations of linguist-computer interactions mentioned above: they are “unfamiliar” in the sense Raskin describes: there are fissures between the way programs display and model documentary data in software interfaces, and the way that linguists are accustomed to thinking about and working with data in non-software-based contexts. If we are to create tools which are helpful for working documentary linguists, then both the nature and structure of the data which linguists record *and* the ways in which linguists go about collecting and recording such data — that is to say, their workflows — must be taken into account.

1.2 Dataflows, Workflows, and Components

I propose to characterize the “lifecycles” of data during documentation projects in terms of *dataflows*: abstract conceptualizations of data transformations from an input state to some modified output state (see Chapter 2). As for the nature of the actual work taken to “carry out” a data flow, we will speak of *workflows* in a general sense — the series of steps linguists take to either collect or record data. Note that these steps may be described in both “real-world” and “digital” terms: for example, sorting a fileslip dictionary “physically” is no less a “workflow” in this sense than using an algorithm to sort a digital dictionary (although of course, here we will mostly focus on the digital variety of workflows). Chapters 3 and 4 will address the details of designing and implementing such tools.

As for actual implementation of user interfaces for carrying out dataflows via workflows, I describe a prototype software library built on the notion of *components*. I have design this library with the goal that it might serve as a flexible basis for implementing a variety of new software tools for language documentation, as opposed to a single “kitchen-sink” application which attempts to meet all needs. The library, called `docling.js`, is based on an approach to software design which is compositional: individual “pieces” of a user interface may be composed into more complex interfaces called *components* or *web components*. Each such component will be defined in terms of (1) the data it takes as input, (2) the data it helps the linguist to produce as output, and (3) the concrete steps the user takes to convert the input into the output. We will refer to the specific definition of transformation of input data to output data as a *dataflow* (see Chapter 2), and the series of steps for

completing a dataflow as a *workflow*.

As we will see in Chapter 4, the example user interfaces support various kinds of workflows, both “traditional” fieldwork methodologies as described in the documentation literature, and more novel, context-specific working contexts. In order to demonstrate briefly what this kind of implementation looks like, we will now look at a small sample component which is designed to help complete a fundamental task in documentary linguistics: eliciting a word list. As we shall see in Chapter 4 on interfaces for carrying out workflows, there are many possible paths to completing the task of eliciting a wordlist (indeed, the notion of “workflow” is intended to assist in articulating the nature of a particular workflow in a detailed manner).

1.3.1 Some sample applications

In the following example, we will begin with a list of prompts, in the fashion of a Swadesh list. As a minimal example, we will imagine that the linguists’ goal is to carry out the elicitation of the numbers ‘one’, ‘two’, and ‘three’. The three stages here are intended as an introductory overview — step 1 will be further detailed in Chapter 2, and steps 2 and 3 in Chapters 3 and 4.

1. **Analyze dataflow** - Specify in detail what changes are to be carried out on specific data types in order to transform input data into output data (this can be visualized with object diagrams).
2. **Analyze workflow** - Specify a series of concrete steps which, if taken, will carry out the dataflow. Note that there are many possible workflows which

can result in a particular dataflow.

3. **Implement a user interface** - Just as there are many workflows which can implement a dataflow, there are many user interface designs, or even combinations of multiple user interfaces, which can be used to carry out a particular workflow. We shall see more examples of this rather general idea below, and in Chapter 4.

This three-way analysis of dataflow, workflow, and implementation is useful for describing the “lifecycle” of data as it progresses through documentation work. In the diagram below, the dataflow and workflow definitions are followed by a simple, implemented application which is functional. The user should try filling in the words for one through three in a language of their choice. The data tabulation will update when the user presses the *Enter* key in any of the three form fields after filling in the forms.

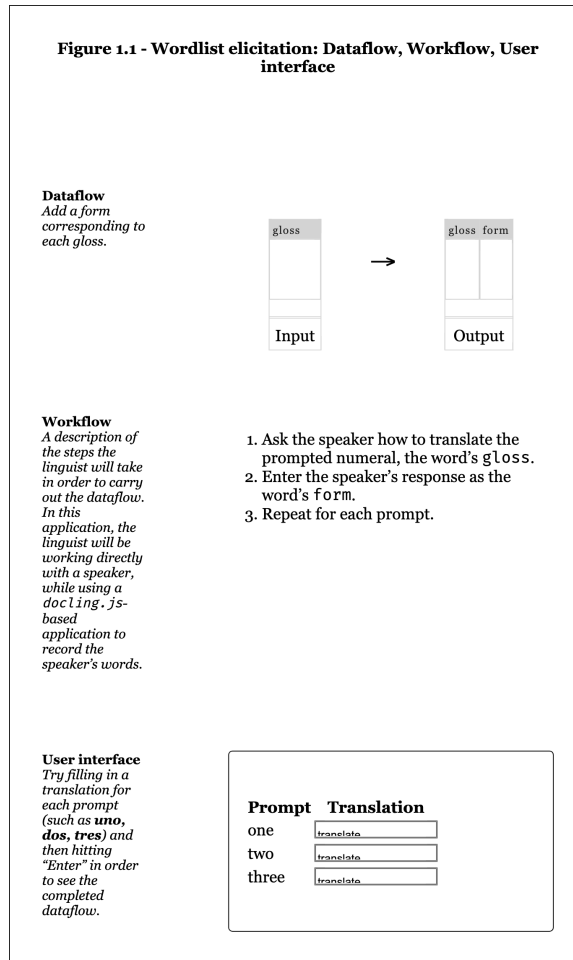
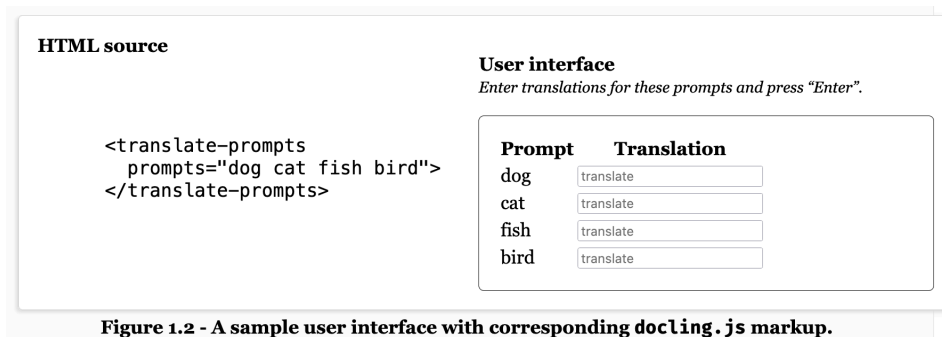


Figure 1.1 - Wordlist elicitation: Dataflow, Workflow, User interface

As a brief preview of the implementation details of this `docling.js` application, consider that one way that this application may be added to an HTML page is through the use of one of the “custom HTML elements” which constitute the `docling.js` library:

```
<translate-prompts prompts="one two three"></translate-prompts>
```

We will see later in detail what custom elements are and how they may be used, but it is worth noting that by simply changing the value of the text associated with the `prompts` attribute of this tag, an instance of this application may be created with a different set of prompts (here, a few animal terms):



In this way, `docling.js` creates an HTML- or “markup-based” approach to *using* applications for language documentation. This is useful because it means that the details of how the interactive elements are actually programmed (which are written in the Javascript programming language) can be made independent from the way in which those interactive elements are used in a web application. Specifications and implementations of examples from many domains of language documentation will be detailed in a similar way in Part II: these include but are not limited to custom elements for various familiar documentary structures such as interlinear texts, phoneme charts, glossaries, grammatical category listings, and so forth.

We should also pause here to point out that the use of *every* user interface brings with it ethical concerns, just as every archival deposit does. The elicitation of a wordlist may seem mundane, but any application for language documentation must be evaluated as to appropriateness within the context of a particular documentation

context. For example, although the elicitation of small numerals may seem fairly uncontroversial, it is by no means certain that numerals are in extensive use in every language, and the discussion of that fact could be a matter of ethical complexity. Members of a team must decide collaboratively whether a particular workflow and interface is appropriate for a given project, and they must also be ready to stop using one if it should prove to be problematic. Importantly, it must be recalled at every step of application design and use that “appropriateness” cannot be programmed or automated.

1.4.1 ELAN and time-alignment

One of the most successful software applications used in language documentation is the EUDICO Linguistic Annotator, more commonly known as ELAN ([Brugman and Russel 2004](#)). ELAN is described as “an annotation tool that allows you to create, edit, visualize and search annotations for video and audio data”. The tool has lived up to this description, and has been used in the production of a tremendous range of time-aligned transcription projects in many human languages. It has, after all, been the key tool in the production of massive amounts of time-aligned transcriptions in many languages and genres.¹ In its most basic usage², the ELAN time-alignment workflow consists of (1) loading an audio (or video) file, (2) viewing a waveform rendering of the audio signal, (3) selecting a subset of the waveform and listening to it, and (4) transcribing the selection. This process is then repeated for the remainder of the recorded material.

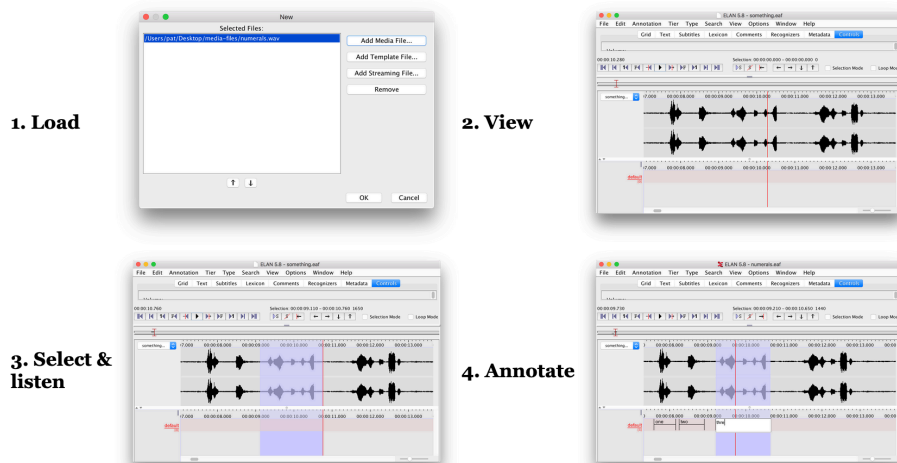


Figure 1.3 - Typical workflow for annotating audio with ELAN

We may visualize this process in the following flowchart:

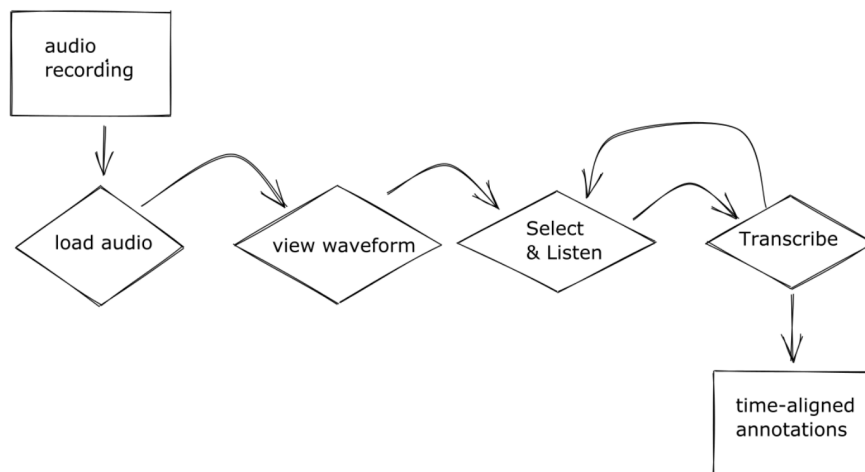


Figure 1.4 - Elan transcription workflow

Let us consider an actual example of the inputs and outputs of the ELAN workflow diagrammed above. The input is an audio recording, in this case an excerpt from a 1966 recording of a word list in Coatzospan Mixtec.

Having carried out the steps described in the workflow diagram above, the

linguist might produce an output which consists of a time-aligned transcription which contains (at least) data of the following kind:

Note that this dataflow treats audio as an *input*: it is assumed that a recording has been created via some *other* workflow. (Indeed, the example audio above is excerpted from a recording available from the UCLA Phonology Archive.)

In time-alignment tasks of this kind, where the linguist begins with a complete recording and proceeds to incrementally transcribe it via the waveform selection workflow, ELAN has proven to be a very useful application. But it is problematic, however, to equate ELAN with *every* type of workflow that results in time-aligned transcriptions of a media source simply because ELAN provides one effective means of doing so.³

This workflow may be used for transcribing any kind of speech genre. In the Coatzospan Mixtec example above, the recording consisted of a pre-defined word list (also known as a *protocol* or *scheduled elicitation*, (see [Samarin 1967, 108](#)). Wordlist elicitation often has a fairly repetitive structure, with each “prompt” being recited in the contact language, and then the word in the language of study being recited one or more times. (It is because of this repetitive structure that linguists recording word lists often go to great lengths to try to avoid the side-effect of recitation known as “list intonation” ([Ladefoged 2003](#)).) In such a rigidly-structured types of speech event, it seems reasonable that a distinct application, with a distinct user interface for aligning transcriptions and recordings, might be useful. In the next section we consider an example of such an application.

1.5.1 An alternative approach to time-aligned lexical data: an iterative recording workflow

Despite the success of the ELAN workflow as described above, it is worth considering alternative ways to interact with time-aligned data. In particular, we may consider a scenario where the audio recording itself is *itself* part of the time-aligned transcription workflow, not the preliminary step.

As we shall see in more detail in later chapters, the web platform’s capabilities include recording from the computer’s audio input. In some circumstances, recording directly onto a laptop is a feasible workflow. Such is the case for the common task of recording and transcribing a “scheduled” word list. It is interesting to imagine what an application which took advantage of this “live recording” ability might look like. In one simple design, we might imagine an application which contains the following ingredients:

Input data

1. A list of prompts.

Output data

1. A time-aligned transcript.
2. A single audio recording of the whole word list.

Workflow

1. An “Enable microphone” button — (Accessing the user’s microphone or

webcam, of course, requires explicit permission from the user)

2. For each “prompt” in the list of prompts:
 1. The prompt itself
 2. A “press-to-record” button, which marks the moment it is pressed down (the moment a segment begins) and when it is released (the moment a segment ends). Recitations may be re-recorded if the recording is not suitable.
 3. A “play” button to review the recorded form
 4. An input to transcribe the recorded form
3. A button to download the combined audio recordings.
4. A button to download the completed time-aligned transcription.

Summarizing as a flowchart:

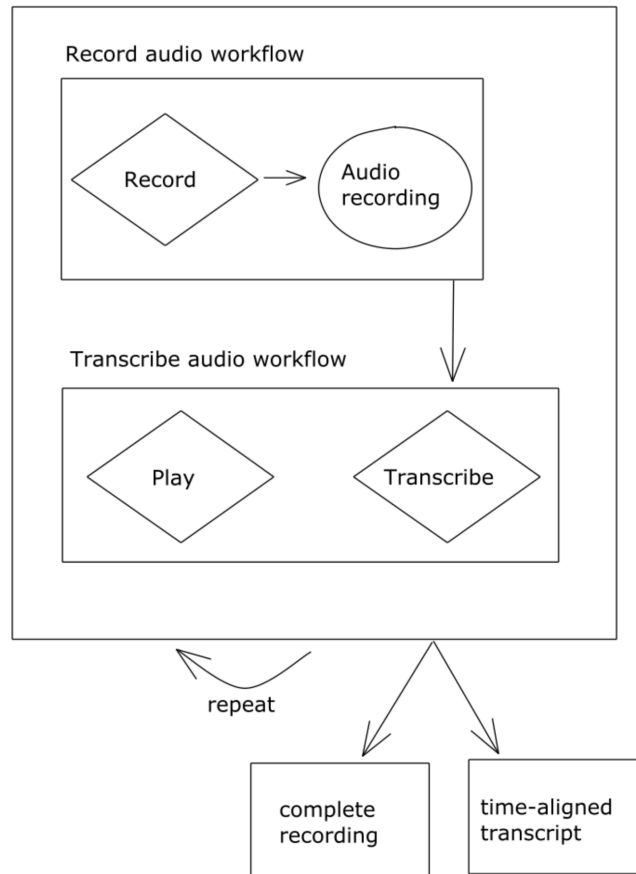


Figure 1.5 - Iterative Recording Workflow

Note in particular that this workflow consists of two smaller workflows. The particular components which have been implemented for these workflows (created from buttons, text boxes, and so forth) could be reused in other contexts, as individual components or as subcomponents of other compound applications. It is also useful that this approach makes the user's hard-drive the sole location for managing recorded files, as opposed to navigating baroque user interfaces on recording devices themselves.

Note that only one of the two outputs created here — the time-aligned transcript

— could be created through the ELAN workflow: the wordlist would simply be recorded and annotated in the same manner that any narrative might be. Because of the incremental style of recording, and because ELAN does not support recording, the production of a “data-only” recording is a new type of output. Interposing a recording step into the workflow obviates the time-consuming sequence of waveform navigation, selection, and annotation.

Here is a working implementation of this application. Help text is available under the help drop-down. The reader is encouraged to try recording translations of the numbers from one to three in a language of their choice, and transcribe their own recitations, and then export the combined audio and time-aligned transcript. (The data tabulation on the right is added for convenience, to demonstrate how the application is keeping track of the user’s work.)

Iterative recording interface

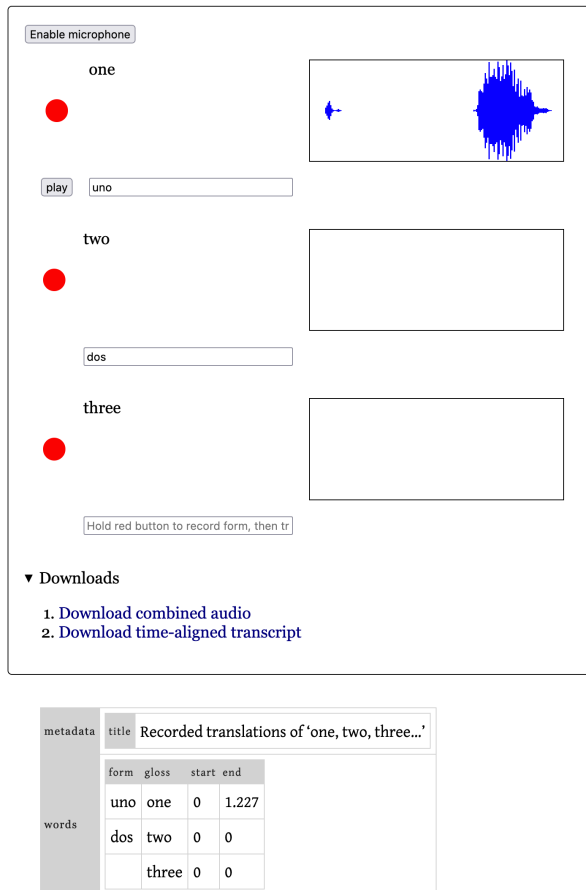


Figure 1.6 - Iterative recording interface

This interface will not win any design awards, but its simplicity does not negate its utility: unlike the ELAN workflow described above, there is no interruption in the workflow between recording and transcribing.⁴ It is not difficult to imagine other extensions of this basic interface: an import mechanism for existing prompt lists, supporting multiple recitations of a single word, and so forth. It is precisely these kinds of relationships between workflows and user interfaces that this work will explore in depth, for a range of documentary workflows that run the gamut of

analytical levels, from phonemes to corpora.

1.6.1 From documents to applications: A functional overview of HTML and the web platform

Recalling that the actual user interface of the sample component above was ultimately constructed out of controls such as buttons, text inputs, and audio player interfaces which are familiar from users' experience using the web, it is worth familiarizing ourselves a bit more with exactly what kinds of default controls are built into standard HTML. There is no need to reinvent the wheel when it comes to such fundamental user interface controls. Indeed, few if any software platforms have been as thoroughly vetted in terms of cross-platform support and accessibility to users of all kinds as the web platform itself.

In this section, then, we will take a whirlwind tour through HTML. That we should plunge immediately into this topic might seem several steps removed from the field of language documentation, but we ask the reader's patience as we lay out the fundamental examples of HTML elements and how they work. It isn't necessary to internalize every detail here at first. What is more important is to understand a general trend which has emerged over the long history of HTML: originally, it was primarily a tool for encoding ("marking up") structured documents in the same way one might compose a document in a word-processing application. However, the inclusion of the *hyperlink* (or "anchor tag") was the first element that *responded* to user activity. This notion of an interactive element proved to be the first of many, as the various *form* elements such as the select element or the ☐ checkbox

presaged more interactivity in the web platform. We will examine this trajectory from static documents to interactive applications more deeply in Chapter 3. In the current section we will simply introduce some default HTML elements as a kind of core catalog of “building blocks” in later descriptions for the custom components which will form the core of our approach, as implemented in the `docling.js` library.

More examples of this growing interactivity are displayed below, and it is in light of this trajectory that the emergence of the newest stage in the development of HTML becomes meaningful: the “custom” element, whose design and behavior is defined entirely by a programmer. Thus we will briefly introduce the topic which will be the main subject of Part II and the `docling` library: a series of custom elements which fit into the web platform in the same way as standard elements, but which are specific to our area of interest, language documentation. Thus, where standard HTML elements include very generic tags such as `<p>` for “paragraph” or `<table>` for a table of data, the custom elements which make up `docling`, such as `<ipa-chart>`, `<text-view>` (for viewing interlinear texts), `<word-view>` (for viewing individual words), and so forth. (Note that non-standard HTML “custom” elements are easily distinguished from standard elements by the presence of a hyphens in their tag names)

HTML stands for *Hypertext Markup Language*. It was originally designed (by Tim Berners-Lee) in 1990, as a tool for exchanging technical documents between particle physicists. That HTML was intended to encode documents is quite apparent if one inspects a few of its *elements*. For example, the `<p>` tag stands for *paragraph*,

and sets its content into a wrapped paragraph with some marginal whitespace above and below the content. The diagram below demonstrates what an HTML tag looks like.

HTML is just a text format; inside an HTML file (which essentially is just a plain text file with the `html` suffix instead of `.txt`), one will find elements with opening and closing “tags” such as the following:

Figure 1.7 - A sample HTML tag.



The term “element” refers to an opening “tag” (such as `<p>`), content, and a closing tag (`</p>`). (As we shall see below, opening tags may also contain *attributes* with names and values.) Our exploration of HTML will not go far beyond these basic facts: the current work is *not* a complete tutorial on writing HTML — here we are merely providing some examples of the kinds of elements that the HTML language includes by default.^{[5](#)}

HTML text	Rendered HTML
<code><p>This is a paragraph, which contains some text.</p></code>	This is a paragraph, which contains some text.
<code>Em (emphasis) elements are typically italicized.</code>	<i>Em (emphasis) elements are typically italicized.</i>
<code>The “strong” element is typically rendered in bold</code>	The “strong” element is typically rendered in bold

Figure 1.8 - Some basic textual elements

It is a crucial capability of HTML that such tags can be *nested* within each other. One simple way to see this is to observe how one may produce not only **bold** or *italic* text, but also text which is both ***bold and italic***:

HTML text	Rendered HTML
<code><p>You can nest bold text within emphasized text thusly.</p></code>	You can nest <i>bold text</i> within <i>emphasized text</i> thusly.
<code><p>Or, vice versa, nest inside bold text with italic text as here.</p></code>	Or, vice versa, nest inside bold text <i>with italic text</i> as here.

Figure 1.9 - Nesting elements

This feature turns out to be one of the most powerful in HTML, because it enables documents to be built up in a structured way. The `<table>` element, for instance, is defined in terms of a set of “sub-elements” (which only occur within `<table>`s) indicating rows and cells. For instance, a simple consonant inventory can

be rendered with a table as follows:

HTML text	Rendered HTML												
<pre><table> <thead> <tr> <th></th> <th>bilabial</th> <th>alveolar</th> <th>velar</th> </tr> </thead> <tbody> <tr> <th>plosive</th> <td>p</td> <td>t</td> <td>k</td> </tr> <tr> <th>nasal</th> <td>m</td> <td>n</td> <td>ŋ</td> </tr> </tbody> </table></pre>	<table><tr><th></th><th>bilabial</th><th>alveolar</th><th>velar</th></tr><tr><th>plosive</th><td>p</td><td>t</td><td>k</td></tr><tr><th>nasal</th><td>m</td><td>n</td><td>ŋ</td></tr></table>		bilabial	alveolar	velar	plosive	p	t	k	nasal	m	n	ŋ
	bilabial	alveolar	velar										
plosive	p	t	k										
nasal	m	n	ŋ										

Figure 1.10 - A simple consonant inventory in a table element.

The most consequential feature of early HTML, of course, was *hyperlinking*. Note that the `<a>` (for “anchor”) element contains an *attribute* with the name `href`. This stands for “hypertext reference”, and its *value* is the link to which the browser should go if the user clicks the link. (Note that the *content* of the link, “*Click here to check out WALS*” is distinct from the value of the `href` attribute.)

HTML text	Rendered HTML
<pre> Click here to check out WALS </pre>	<p>Click here to check out WALS</p>

Figure 1.11 - A hyperlink, or “anchor tag”.

Hyperlinks were mentioned above. It is the `<a>` tag which implements that notion, and it is worth investigating its structure and functionality briefly. Firstly, it contains an *attribute* with the name `href`. That rather obscure acronym stands for “hypertext reference” — this is the substance of the “link”, as it is the destination to which the user’s browser will be redirected upon clicking the text. The *value* of the `href` attribute is usually a URL. In this case, the link points to `https://wals.info`. Again, note that this element is inherently interactive: the link is only “followed” by the browser when the *user* chooses to click it. This kind of behavior is at the heart of the interactivity of the web platform.

In the first version of HTML, only the anchor tag was interactive in this way. The user could only read and follow links. But as the HTML standard has evolved, new elements have been added, adding support for new kinds of interactivity. One particularly important development was support for text input. In the illustration below, a dropdown menu allows the user to select one of three options (in this case, simple word class labels). This control is appropriate where the user should be able to select one of a closed set of options. Each alternative is nested inside the `<select>` element in its own `<option>` element.

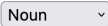
HTML text	Rendered HTML
<pre><select name="pos-select"> <option value="noun">Noun</option> <option value="verb">Verb</option> <option value="adjective">Adjective</option> </select></pre>	

Figure 1.12 - Part-of-speech selection implemented as a drop-down menu using the `<select>` tag.

Note that the HTML standard sometimes offers more than one way to accomplish

the same “data operation”. For instance, the interface below — referred to as “radio buttons” after their analogy to the preset buttons on old car radios — accomplishes precisely the same goal as the previous control. (Which user interface do you prefer?)

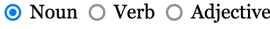
HTML text	Rendered HTML
<pre><div> <input type="radio" name="pos-radio" value="noun" checked=""> Noun <input type="radio" name="pos-radio" value="verb"> Verb <input type="radio" name="pos-radio" value="adjective"> Adjective </div></pre>	

Figure 1.13 - Part-of-speech selection implemented radio buttons with `<input type=radio>`

Below are two final elements for text entry, which are of course of great importance for language documentation, where entering text is a ubiquitous (and unavoidable!) task. `<input>` and `<textarea>` tags differ in that the former does not allow the user to insert a line break into the text, whereas the latter does. In fact, there are reasons for both kinds of inputs.


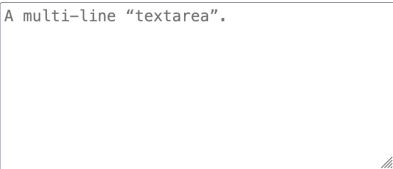
HTML text	Rendered HTML
<pre><input type="text" placeholder="A single-line input."></pre>	
<pre><textarea rows="8" cols="40" placeholder="A multi-line 'textarea'."></textarea></pre>	

Figure 1.14 - Single-line `<input>` and multi-line `<textarea>` text entry controls

As the HTML standard has evolved, elements which can handle more complicated kinds of interactivity have been added. Most notably for documentary

linguists, media elements have been incorporated into the standard. The simplest of these is support for images via the `` tag. The `src` attribute of an `` tag below specifies an image file, which is then imported into the browser and rendered in place.

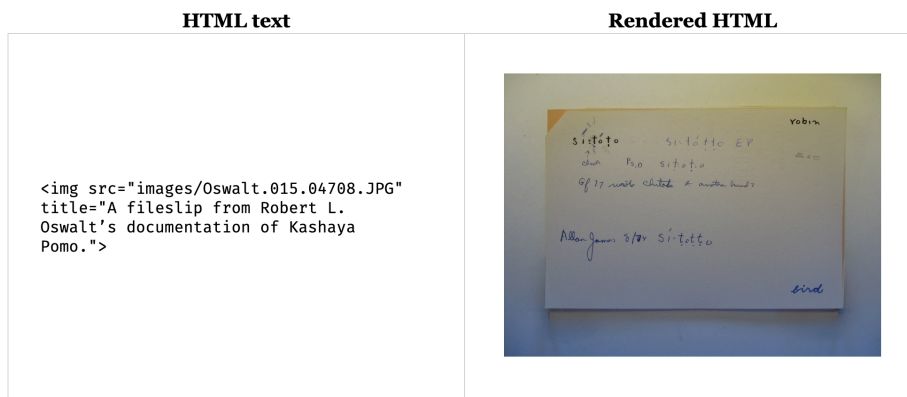


Figure 1.15 - An embedded image.

Other kinds of file may also be embedded, for instance, PDFs:

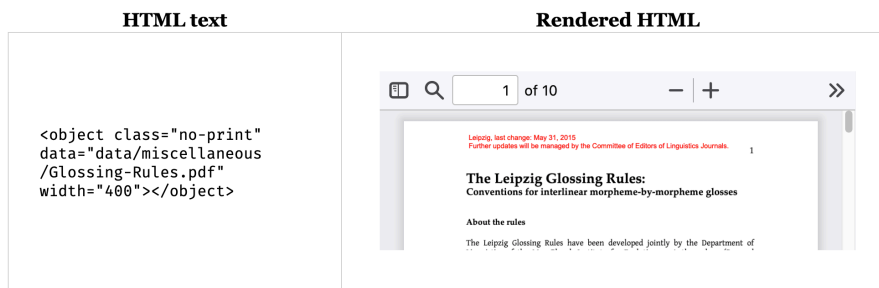


Figure 1.16 - Including a basic PDF interface

Most exciting of these kinds of embedded content for linguists and others involved in language documentation are embedded media. The addition of `<video>` and `<audio>` tags in version 5 of the HTML standard have brought new levels of interactivity to the web.



HTML text	Rendered HTML
<pre><video src="data/languages /hiligaynon/corpus /education_in_jaro /education_in_jaro.webm" controls="" width="400"> </video></pre>	
<pre><audio src="./data/languages /hiligaynon/corpus /hil_wordlist_1965/hil_word- list_1965_01.wav" controls=""></audio></pre>	

Figure 1.17 - Media elements

As we shall see later, the default presentation of `<audio>` and `<video>` tags may be modified via various tag attributes. It is also possible to programmatically adjust other details of media playback, such as playback rate, via Javascript. Below, the same two media elements have been modified programmatically to slow down the playback rate. (This functionality might be helpful for trying to describe particularly fast speech.) Another new functionality of media tags is the so-called *media fragment* syntax, which enables an HTML author to specify a “clip” of the entire recording to play by default. This is done by appending start and stop types to the value of the `src` (source) attribute. Note that media fragments can be controlled directly from page markup, as opposed to requiring a custom Javascript program.



HTML text	Rendered HTML
<pre><video id="slow-jaro" src="data/languages/hiligaynon /corpus/education_in_jaro /education_in_jaro.webm#t=30,40" controls="" width="400"></video></pre>	
<pre><audio id="slow-hil-wordlist" src="./data/languages/hiligaynon /corpus/hil_wordlist_1965 /hil_word- list_1965_01.wav#t=30,40" controls=""></audio></pre>	

Figure 1.18 - Media elements with fragment references

1.7 Custom elements for language documentation: `docling.js`

As we have seen, some of the default HTML elements have a significant amount of interactivity encapsulated into their definition. In order to create a usable video player, for instance, one need only insert a `<video>` tag into a web page, and specify the file to be played. But the document-oriented origins of HTML remain. Standard HTML elements are still very “generic” in the sense that they are in no way linked to any particular domain of knowledge aside from very generic document structure. Consequently, the set of default elements is intended to support the creation of rich, interactive documents, but the HTML standard itself has never been designed with any particular *kind* of content in mind.



**Figure 1.19 - The Wugly
Docling, our fearless
mascot**

A new development in the web standard, Web Components, has provided tools to for customizing HTML itself. (WHATWG) ([2020](#)) This technology is the basis of the current work: specifically, the Custom Element sub-standard of the Web Components defines a way for authors to create new HTML elements — distinguishable from standard elements in that their tag name must contain at least one hyphen — which can be programmed in arbitrary ways. The current work is in large part an explication of a library of such custom elements called `docling.js` (for “documentary linguistics with Javascript” — many Javascript-based libraries are named in this way). Each element will be explained through the dataflow/workflow/implementation lens. In some cases we will analyze the dataflows and workflows of existing software (such as ELAN, above), and develop a new custom element intended to fill a gap in current practice. In other cases we will be generating elements which are intended to be composed into larger applications. For instance, by building on the implementation of an `<orthography-view>` element which displays a set of orthographies for a particular language, we can create another element, `<transliteration-editor>`, which refers to the `<orthography-view>` element, but which aids in a distinct task: transliterating from one orthography to another.

At this point I wish only to introduce the concept of web components — much more will be said about why web components are a good match for a community effort for creating language documentation software in Chapters 3 and 4.

1.8 On motivations and voice

While this work is not a complete tutorial on programming, it does amount to a thorough introduction to how web technologies can be used to construct useful applications for linguistic fieldwork. One might question the utility of writing “about” programming without explicitly teaching programming. But in my view, attempting to train all participants who create and work with digital language documentation materials to become programmers is not a realistic, or even desirable, goal. But the community as a whole seems to be in a kind of holding pattern insofar as we are relying exclusively on existing software tools, some of which have changed little in a decade. Few new user-facing applications for the fundamental tasks of transcription, time alignment, morphological analysis, and generation of reference and pedagogical materials for language revitalization and learning have emerged in the recent years. Worse, the existing tools were not designed to be interoperable, and thus, by necessity linguists must cobble together work-around workflows involving complex input/output steps. And even when new tools do emerge, it seems to be accepted as a given that they will only expand the current complex of tools. Consider for instance this diagram delimiting how one tool (Audiamus) was expected to fit into the existing software network:

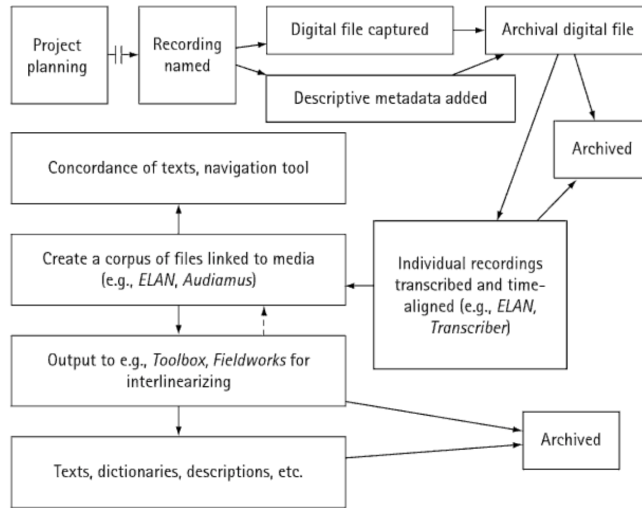


Figure 1.20 - Berez and Thieberger:
Workflow for creating well-formed
linguistic data ([Berez and Thieberger 2011](#))

Such a complicated combination of resources make this workflow difficult to maintain. Four distinct applications are explicitly mentioned, and presumably others would be necessary for other steps. While all of the steps addressed by this diagram are important and integral steps in language documentation, the notion that they should be parceled out to individual applications, each with distinct platform requirements and learning curves, is not a sustainable approach for the future.

A simple armchair experiment is sufficient to show that this state of affairs is anomalous. Let us imagine that some organization, perhaps the LSA, somehow found the means to fund a massive new software project with infinite programming resources. This magical project would handle all of the kinds of documentary data

and usage that we would want to support. In this scenario, it seems unlikely that such a workflow, with its baroque input/export procedures, would be recreated. Such systems may be a testament to the dedication and ingenuity of their creators, but they are nonetheless products of historical accident. Surely, that hypothetical system would be designed in such a way that all layers of linguistic analysis would be handled in a similar fashion: working on morphological analysis, for instance, would not be siloed into a separate application from an application for carrying out time alignment.

But we do not need massive funding and an army of programmers to get started down this road. And that alternative possibility is what I am trying to address in this work. If nothing else, I hope to show that we can imagine new ways of doing and using documentation that are built on a coherent underlying representation of documentary data — a representation which should be familiar to working linguists. My intent here is to design and implement a modular approach to building software for language documentation. I have striven to design this system in a way that there are many “points of entry”: a user who is interested only in making use of a completed corpus should be able to view content in a web browser as simple HTML pages. A user who is *creating* documentation should be able to understand how to use basic user interfaces (such as a dictionary editor or interlinear text editor) with a minimum of training. And those who wish to delve into customizing or even creating applications should be able to work at any linguistic level: the actual implementation of an interface that deals with corpus-level analysis is very similar in terms of code implementation to one which is designed to work at a much lower level, such as annotating phonetic features of a single phone. Data can be passed from one to the

other. And *both* work in the very same computing environment, the most universal computing environment yet developed: the Web Platform. Our reliance on the “software status quo” has engendered a rather insidious complacency: as far as software is concerned, the complacency says, we should just make do with what we have. This work is my attempt to push back against that notion. We should not just make do, we should strive to create a system which is at once coherent and flexible, future-proof and extensible.

And it is this spirit of universality that I must mention two unusual choices I have made in terms of the style of this work. Firstly, the reader may have already noticed that with the exception of the current section, I write in the first-person plural. That choice may seem anachronistic or even pretentious (the royal we?), but after several attempts to recast the work in an alternative style, I am convinced that “we”, “us”, and “our” are the right pronouns here. I want to bring others into the fold, in the sense that all linguists should see that they can play a role in how they make use of their computers to do their work. Writing a dissertation can be a lonely endeavor — what has carried me through has been the motivation that my work may become a small step in a path toward a larger, community effort.

The second choice has to do with how I use the term “linguist”: namely, I assign it a meaning which is very broad. While there has been commendable and considerable discussion of the notion of collaborative linguistic fieldwork in recent years (e.g., Yamada ([2007](#))), our terminology is in some ways becoming a victim of its own success. After all, language documentation has a long history of blurring the lines between “linguists” and “speakers”: just to cite the North American context, Parker

McKenzie, Paul Platero, and Ella Deloria were all linguists as well as “speakers”. Rather than proliferate labels for all the infinite possible combinations of roles (which can also change for an individual from project to project or year to year), I simply refer to all participants as linguists.

1.9 Outline in brief

Below is a brief overview of the contents of this work.

[Introduction](#)

Background context regarding linguists and their data. Brief introduction to HTML markup, the Web Platform, and a library of custom HTML elements designed specifically for use in language documentation. Organization of this work.

[Data types and dataflows](#)

A data type is a machine-readable means of representing information in a computer. In this chapter, we define the term “data types”, and explain their relationship to workflows and components. We also give a bird’s-eye view of the full hierarchy of data types developed herein.

[Viewing documentation](#)

We begin by considering “Views” — custom HTML elements which display

documentary data structures .

Components in fieldwork

Having seen how documentary data structures can be displayed, we consider the larger task of designing user interfaces for creating and editing documentary data. We demonstrate the importance of context to documentation workflows, and

Avenues ahead

Thoughts about possible reuse of these techniques, including language revitalization, language pedagogy, and further analysis for typology.

2. Data types and dataflows: Building a Boasian Database

2.1 Toward a Boasian database

2.2.1 Designing a documentary database

The term *hypertext* hasn't aged particularly well. The futuristic connotations that generated so much enthusiasm for the web in the eighties — when many believed that ubiquitous access to knowledge would be an unbridled boon to humanity — must now be at the very least counterbalanced by all of the misinformation and antisocial uses to which the web has been put. It is very much a double-edged sword. And yet, the basic concept behind hypertext — the idea that knowledge is not necessarily always best “consumed” linearly, like a novel, but rather as a network of linked, interrelated parts which may be traversed in non-linear fashion — remains a compelling one. And, to readers, writers, and users of language documentation, reading in a non-linear fashion is very familiar.

Linguists often jump around their combined resources — such as grammars, dictionaries, corpora — following cross-references of many kinds. Indeed, one might be so bold as to argue that language documentation, especially, has always had a “hypertextual” character. This is because documentation is constructed from information of varied types, any particular datum of which must be evaluated — indeed, can *only* be evaluated — in the light of how that particular datum relates to *other* data. As Firth famously put it ([1957](#)), “you shall know a word by the company it

keeps.” But for the working documentary linguist, Firth’s mandate does not address the crucial detail: if we are to treat our documentation as a model of relationships between words and other linguistic units, *how* are we to actually go about managing the data that represents those relationships?

For our goals, put bluntly, this question amounts to asking “how should we put our documentation into a computer?” The term “well-structured data” is often thrown about, but what does it mean, exactly, to say that data is “well-structured”, or even “structured” at all? Exactly what kinds of information are we to include? How is that data to be encoded in a machine-readable way? And what kinds of utility will such effort to encode documentary information ultimately provide? We may summarize these questions with one overarching question: **How should we design a documentary database?**

Designing a database is no small undertaking — there are professionals whose entire careers are dedicated to the task. Fortunately, however, we do not have to start from scratch. While the full arc of the long history of language documentation was mostly pre-digital, thinking about *types* of data is in no way dependent on the use of a computer. As we shall see below, many of the notations and nomenclature developed by the field, even in a pre-digital era, can be conceptualized as a kind of database structure. Indeed, one of the most familiar frameworks for thinking about data in documentary linguistics — the *Boasian trilogy* — can be thought of as such a design, at least in broad strokes. The trilogy consists of the combination of three “data types”: the corpus or text collection, the grammar, and the dictionary. But as Woodbury ([2011](#)) points out, Boas ([1917](#)) was more interested in documentary data

as a network than as a mere list of types:

“All three were interrelated parts of a documentary whole, treating, in different ways, overlapping empirical domains; and it would be a mistake to project from any one of these a specific theoretical domain or level of analysis.” ([Woodbury 2011](#))

The defining idea of the trilogy is that its parts are inherently interrelated, not simply that they are each required individually: data from each of the three documentary types must be understood via reference to the other two. As Woodbury emphasizes, the key insight behind the notion of the Boasian trilogy is not that we should merely list three distinct, useful types of documentary data, it is the recognition that those three data types *are inseparable*. In the print domain, such interrelationships between the three main documentation types is approximated with the familiar textual mechanism of the cross-reference: by means of arrows, as it were, from one point in the stream of text to another point in that stream.

In the remainder of this section, we will investigate the nature of such cross-references in the print and digital domains, proceeding to the notion that neither cross-references nor hyperlinks are sufficient in and of themselves to digitize the defining interrelatedness of the Boasian trilogy. Modeling interrelationships in data is, of course, a task at which computers can be made to excel — indeed, a database may be understood (at least in part) as a network or graph of interrelated information which enables “manipulation and arrangement” of many kinds. But the construction of any database depends on clear definitions of the internal structure of each of the *data types* with which it is intended to capture. We will describe a

particular machine-readable “object notation” known as JSON that can be used to encode structured data in a machine-readable way. In the third section, we will outline a core catalog of data type representations which emphasize rather than obscure such interrelatedness.

This shift toward conceptualizing documentary data in terms of structured computational objects — as opposed to textual content with textual cross-referencing — offers many benefits to the working documentarian as well as to linguists in general. We shall see how the object model enables *hierarchy* to be encoded in a useful way: thus, words may be represented as “containing” morphemes, words may in turn be collected into a lexical analysis of a higher-level object with translations and transcriptions — a sentence. Sentence objects are contained in Text objects, and at a higher level still, Texts may be collected into a Corpus. The list of unique individual words collected from texts are then collected into a *dictionary* (or *lexicon* — on the distinction, see [the section below](#)). And finally, we will see how grammatical categories may be treated as an enumerable data types in their own right — that representation, while simple, goes a long way toward making claims about the distribution of grammatical categories within usage more accountable to the corpus. Finally, in the last section, we consider another useful product of this way of conceptualizing our documentary data: the *dataflow*. Having described a specific method of encoding documentary data in sections 1-3, section 4 shows how particular configurations of data may be thought of as inputs and outputs of particular documentary tasks. Thus, as mentioned in the previous chapter, a word elicitation task may be conceptualized as having an input consisting of a list of glosses (“prompts”), and an output consisting of “words” with glosses and forms.

This notion of dataflows is both abstract and generic: it does not describe the actual *steps* which must be taken to transform an input into an output. But this abstractness is by design: by characterizing such transformations of documentary data in an abstract fashion, we will be in a position to experiment with many kinds of *applications* — implemented user interfaces which enable users to carry out particular steps which “complete” a dataflow.

2.3.1 A shift in viewpoint: from streams of text to databases of objects

Jeffrey Heath’s documentation of the Australian language Nunggubuyu, is a prime example of a thoroughly cross-referenced “print trilogy”. Published as three volumes — *Functional Grammar of Nunggubuyu* -Heath ([1984](#)), *Nunggubuyu Dictionary* -Heath ([1982a](#)), and *Nunggubuyu Myths and Ethnographic Texts* -Heath ([1982b](#)), Heath’s work maps directly onto the three parts of the Boasian Trilogy. But the degree to which all three parts are interrelated through extensive cross-referencing is remarkable. The three volumes cross-reference each other in a deep way, such that the interrelated content of the three volumes functions as a kind of “print hypertext”.

The great value of Heath’s granularly interlinked trilogy was recognized in an interesting reconsideration of Heath’s work (Musgrave and Thieberger 2012, 64). They trace references of many kinds amongst the three volumes. In this section I recreate the references they used, discussing them in the context of our current discussion. In the case of the dictionary, they cite the following entry for the form *dhan^sid!* ‘to chop’:

dhana⁹id! Rf to chop. 16.14.3, 43.4.1, 43.6.4. Associated with verb =lha- ‘to chop’.

Figure 2.1 - Dictionary to Corpus reference: Heath (1982a), as referenced in (Musgrave and Thieberger 2012, 64)

The entry begins with the form itself and then two senses. Each sense contains a word class symbol (here, Rf), and a simple definition *to chop*. I have highlighted the headword, which is also found in the corpus excerpt below. Such example sentences are of course typical of many dictionary entries, but it is the last feature that makes Heath’s approach stand out: following the definition, he gives a *exhaustive* list (namely, “16.14.3, 43.4.1, 43.6.4”) of attestations of that form within the corpus volume (Heath 1982b). It is this last element, which might be said to be “hypertextual”. Heath describes his referential system as follows:

The three-part sequences like “15.4.3” refer to NMET (i.e., my own texts volume) and indicate text number (15), segment number within that text (4), and line of Nunggubuyu text within that segment (3).

The citation scheme Heath describes here cross-references three levels of specificity: an identifying number of a text, a paragraph number (or using Heath’s term “segment”) within that text, and finally a line number within the segment. Note that the entry given by Musgrave and Thieberger is actually rather atypical of Heath’s dictionary, in which many entries contain textual references with a dozen or more such references. Heath was attempting to be comprehensive in the linkage between the three volumes.

But referencing particular lines within a segment is in a sense “extra-linguistic”. It is dependent on the layout characteristics of the text itself. If the typography of the text were changed somehow — say, if the content were re-set in a different font size, or a different typeface, then the entirety of the text would rewrap. Consequently, the references to particular “lines” would have to be recast (an enormous undertaking in a work of this scale). As Musgrave and Thieberger point out, the linkages between dictionary and corpus are closely linked to the physical print layout, which requires the reader to count physical lines in a segment in order to resolve the reference.

43.4 wu=wayama-n^oi-ya-j mari dhan^oid! adaba 0=lhi-n^y
as it proceeded, and chop then it chopped it

o ana-ran^oag, 0-madhari-n^y 0-madhari-n^y g-madhari-n^y
wood it chopped itp nearly

wu-ragar=bayama-ngi mari n^oijan^o wurugu dūlmurg!,
it went along forcefully, and more later run

wini=wilbili-n^y arwagarwar-ala-aj,
they (MDu) flew around on top

It (devil) came along and began to chop down the tr
chopping and chopping. It (tree) was about to crash
then they (two) flew away. (They flew) around up hi

**Figure 2.2 - Corpus: Heath (1982b), as referenced in
(Musgrave and Thieberger 2012, 64)**

Reference may also be directed from the corpus to the grammar, and Musgrave and Thieberger highlight the form *mari* as occurring in the grammar volume with its own set of corpus reference. There is a section of the grammar about this particular particle, as follows:

This particle can combine with other particles, We mentioned /**mari** wurugu/ and /wurugu n⁹a/ in the previous section (it is likely that /n⁹a wurugu/ also occurs), We can cite /n⁹ijan⁹ wurugu/ (cf. next section) ‘again later’ or ‘more later’ 21.9.1, 21.10.1, 33.1.2, **43.4.3 (with preceding /mari/)**, 43.5.2/4, 52.5.2/3, 163.19.2/3, showing this order to be consistent.

Figure 2.3 - Grammar to Corpus reference: Same line referenced in grammar, §4.8 of Heath (1984)

The amount of information encoded in this verbose referencing system is impressive, but using three physical volumes is unwieldy at best. Even the scanned versions of the volumes, now available online, are difficult to traverse. Confronting this state of affairs, which is both rich in information and difficult to use, is frustrating. It is clear that references may point in principle between any two of the grammar, dictionary, and corpus, as diagrammed below:

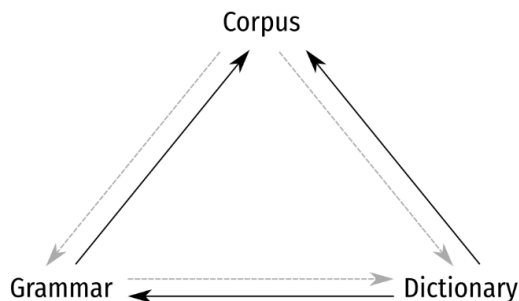


Figure 2.4 - Six possible kinds of references in the Boasian trilogy

There are six possible “directions” of reference among the three parts of the

Boasian trilogy, and we have seen examples of three such references: 1) from the grammar to the corpus; 2) from the dictionary to the grammar; and 3) from the dictionary to the corpus. Referencing of this kind gives the reader confidence that claims made about lexicon, grammar, and usage of Nunggubuyu are being described in a defensible (and checkable) manner.

There is an important issue to keep in mind as we consider how to design a system which can emulate all of these kinds of links: there is a difference between simply “pointing at” from one point in text to another point in text, and actually *modeling* data in such a way that linkages between data objects can be maintained automatically. In the next section we address this idea more deeply, and then proceed to develop a set of concepts and vocabulary for modeling data which we can use to create a “Boasian database”.

2.4.1 Beyond hyperlinks

Heath’s extensive cross-referencing increases the documentary power of his Nunggubuyu documentation greatly: it helps a reader to traverse the non-linear trilogy effectively. Hyperlinks carry out much the same function as cross-references, but in digital form (and without the need to physically manipulate volumes and pages). This “jumping” functionality is undeniably useful in both the print and digital domains, but such links do not *in and of themselves* constitute a “database” in the sense we will employ here. Even given its relatively early date of publication, Heath set high goals for the usefulness of his work: he explicitly states that his goal in creating a print “database” was to make his own work easier to subject to “cross-

examination”. Here, Heath seems to be stating a goal that goes beyond *just* cross-referencing (or, in web terms, hyperlinking). He wants his readers to have access to the “raw data which underlie the analysis”. Heath’s goal is to make it possible for other linguists to carry out original research using his documentation. In other words, he wants his work to be usable as a *database*, and not only as a (readable) *document*. This stance is admirable, and quite remarkable given the year of publication. He explains that his approach:

“...gives a more patient (or more skeptical) reader a feeling for the raw data which underlie the analysis and an opportunity to ‘cross-examine’ the author by going directly to the data. It also encourages readers with highly specialised interests, or with a different theory of language, to discover new patterns which I overlooked or did not have space to discuss.” ([Heath 1984, 5](#))

Taking this dedication to accountability as a starting point, Musgrave and Thieberger describe a sort of “wish list” for a digital linguistic database:

The interrelatedness of the various components discussed above immediately suggests that hypertext would be a better means of presentation and additional benefits could come from making the grammatical description a multimedia object, rather than a text object. Examples could be heard in the original sound recorded by the researcher, or even seen as video clips where such presentation would aid the consumer (for example, where gesture added an important element of meaning to the utterance). In addition to the improved accessibility of the descriptive

information, such presentation would bring the consumer much closer to the primary data, actual language in use, and therefore multimedia language description would increase substantially the standard of accountability in linguistics. However, the standard paper and ink presentation of grammatical description has an established linear format which is not suitable for the new medium.

Thieberger and Musgrave suggest that multimedia references be added to Heath's database, since time-aligned audio or video could enhance the documentation by bringing even more "accountability" to it.

In the following excerpt Musgrave and Theiberger describe their methodology for actually carrying out the digitization:

A small segment of the description of Nunggubuyu is available online at <http://users.monash.edu.au/~smusgrav/Nunggubuyu>. The XML source of these pages was hand-coded and HTML was then generated using search-and-replace in a text editor. Obviously, these procedures are time-consuming and, having established reasonably stable principles for encoding the material, our next priority is to automate the process as much as possible.

The ability to follow links from individual texts (such as Milton [Gabanja] and Heath (2018)) to dictionary entries (such as Milton [Gabanja], Heath, and Musgrave (2018)) is very useful, and the demonstrations of hyperlinked navigability in the online sample are certainly tantalizing. Obviously re-keyboarding the entirety of

Heath’s work would be a mammoth undertaking. Even so, the goals of the digitization project as described in Musgrave and Thieberger (2012), seem to be limited to a process of converting textual cross-references into digital hyperlinks.⁶

But such a complete conversion, even if completed, would miss important functionality that Heath himself foresaw as desirable. Expressing the details of just what sorts of functionalities are possible in a “Boasian database” will constitute the remainder of the present work. In general, however, we may emphasize again the importance of the idea that documentation is *not* best conceived of as a set of three extended texts (grammar, corpus, dictionary) which contain cross-references or hyperlinks to other points in those texts. Rather, we should *model* our data as collections of abstract “objects” which have complex, composite internal structures. We must shift from thinking about our documentation through the lens of a text model, to thinking about it through the lens of an object model.

A clarification of the notion of an “object model” is the goal of the next two sections. We will explore what it means to “represent” or “model” a real-world entity in a database, focusing on a carefully chosen set of attributes of an entity can serve to identify it in a useful way. We will walk through some examples of such data modeling below. We will discuss how *objects* may be understood as a bundle of *attributes*, in turn consisting of *properties* (names) and *values*. We will introduce two ways to conceptualize objects: firstly as an abstract “nested table” visualization, and secondly as a machine-readable data notation called JSON, for JavaScript Object Notation. The visualization and the syntax are directly interchangeable. By thinking of the data captured in documentation in these terms, we shall see how


the data-interrelationships Heath sought to represent as cross-references in his Nunggubuyu print database can be operationalized as an *object database* in a real-world computational environment — the web platform.

2.5.1 What is database design?

Perhaps the most widely known form of database is the spreadsheet. Spreadsheet programs such as Microsoft Excel and OpenOffice Calc are exceedingly common in many kinds of work. The basic steps to creating a spreadsheet seem almost trivial: one writes down headers for each of the columns of interest, and then fills in values for each header in each row. We create such a table for each kind of information we are interested in tracking, and voilà, a database which can be sorted and searched in various useful ways.

Of course, this is a simplification of the process. For one thing, spreadsheets are created for specific purposes, and when someone sits down to create a spreadsheet, he or she is usually already fairly familiar with the *type* of data that they want to record. It is easy to overlook just how many conventionalized “data definitions” we make use of in our daily lives. Consider, for example, a non-linguistic example: cars. Most people are familiar with cars, and interact with them on a regular basis. We drive them, we ride in them, and on occasion we buy them. Let us consider that last kind of interaction. Specifically, let us try to articulate some how we conceptualize *kinds* of “cars” when it comes to buying one. Consider this vintage car advertisement, for instance:

Come in...LOOK AROUND



at these top value used cars


AT
ELLWOOD MOTOR CARS, inc.

1960 VOLKSWAGEN\$1395
Radio and Heater with White Sidewall Tires.	
1959 VOLKSWAGEN\$1195
Heater and White Sidewall Tires.	
1958 VOLKSWAGEN\$1095
Radio and Heater with White Sidewall Tires.	
1960 COMET\$1295
Radio and Heater. A Very Clean Car!	
1960 FALCON\$1195
Radio & Heater. White Sidewall Tires.	
1958 PLYMOUTH\$995
Drivers & Dr. R&L, Automatic Trans., 31,000 miles, Like New in Every Respect!	
1958 BUICK\$1195
Century Convertible WITH Full Power!	

• • •

BANK FINANCING AVAILABLE TRADES ACCEPTED


OPEN 'TIL 7 P.M. MONDAY - FRIDAY



These cars Carry
GUARANTEED WARRANTY
for One Full Year

Ellwood Motor Cars, inc.

630 Bellough Rd.



CL 2-2023

Authorized Dealer

Figure 2.5 - A vintage used car advertisement with seven listings

It is not difficult to reconceptualize the information in this ad as a spreadsheet. A few details need to be reconstructed: the “Comet” was a model from Mercury, the model of Volkswagen was probably a “Bug” and so forth. Indeed, it is interesting to consider the fact that the advertiser felt comfortable leaving out the “model” and “make” on a few of the listings. This probably is reflective of the fact that the average 1960s newspaper reader already knew that a Comet was a model of the Mercury “make”, and that a Volkswagen in the \$1000 range was probably a Bug (and not a van, for instance). We make use of this kind of background knowledge constantly. And it is often precisely that kind of background knowledge that is put to use when

someone creates a new table in spreadsheet: we “just know” — it is part of our knowledge of our world — that the convention for designating a *kind* of car is to specify (at least) its year, make, and model. To this author’s ear, at least, the three attributes are so conventionalized that they even demand a particular grammatical order. “1958 Plymouth Belvedere” sounds grammatical, but any other permutation is not: 1958 Belvedere Plymouth (?), Plymouth 1958 Belvedere (?), etc.

We can express, then, the complete information implied in the advertisement as follows:

year	make	model	price	features
1960	Volkswagen	(Bug)	\$1395	radio, heater, white sidewall tires
1959	Volkswagen	(Bug)	\$1195	heater, white sidewall tires
1958	Volkswagen	(Bug)	\$1095	radio, heater, white sidewall tires
1960	(Mercury)	Comet	\$1295	radio, heater, very clean
1960	(Ford)	Falcon	\$1195	radio, heater, white sidewall tires
1958	Plymouth	Belvedere	\$995	4 door, radio, heater, automatic transmission, 31000 miles, like new
1958	Buick	Century	\$1195	convertible, full power

Figure 2.6 - Car advertisement data reformatted as a spreadsheet

It is this set of attributes of a car that constitute the shared understanding between the seller and the buyer — were that not the case, the seller at Ellwood Motors would not have paid for an ad spot in a newspaper containing just that information. Consider, for instance, the fact that the ad says nothing whatever about color.

We might assume that such a thing is obvious, but is it? When a hypothetical

customer spotted the second-from-last listing and muses, “That’s a loaded Belvedere, I’ll go down to Ellwood Motors and check it out”, what exactly do they expect to see? They expect all of the attributes of the car in the listing to be fulfilled. What is that information? Roughly, it consists of the values in the row of the spreadsheet that contains that listing. After all, that information is all that was exchanged between the seller and the buyer in order to enable a potential sale: the year (1958), the make (Plymouth), the model (Belvedere), a particular price, and a list of features (four doors, a heater, a radio, an automatic transmission, 31K miles and it’s in “like new” condition).

Like any word, these notions have complicated semantic ranges. After all, what exactly is a “make”? Trying to define that in *a priori* terms is every bit as difficult as defining the senses of any polysemous word in a dictionary entry. Consider how readily we can determine what should *not* be included in a database, given our background knowledge of the world. Consider the “2008 Sikorsky S-434”. Those three tokens seem like they could very well be a year/make/model triple, and should therefore fit into the database, just like a “2019 Tesla S3” would. Except that Sikorskies are helicopters, not cars. It is a very difficult problem to explain how humans are able to maintain a workable understanding of how “the world” can be boiled down to heavily conventionalized representations such as the year/make/model triple. And yet, humans do that.

When we create a database, then, more often than not, all we are doing is formalizing such a representation as a list of objects represented in terms of these *identifying attributes*. The car buyer and the seller already share a “working

definition” of cars: at the very least, they share the notions of year, make, and model. There are many shared conventions for representing cars, each of which has its own list of conventionalized identifying attributes: in the US, for instance, the Department of Motor Vehicles (DMV) makes use of the “Vehicle Identification Number” (or VIN). That database doesn’t stop at identifying *kinds* of cars in terms of year/make/model: the *kind* that the DMV’s database tracks is *unique* cars, specific cars with particular accident and ownership histories, and so forth. (Copying a VIN from one car to another is actually considered a criminal enterprise!) The DMV has its own reasons for caring about uniquely identifying cars — law enforcement, taxation, inspections, etc.

Considerations such as these are behind the design of every database, no less in language documentation than any other field. Fortunately, both the conventional data types and their lists of identifying attributes have been fairly well determined as the field has evolved. We just need to write them down and implement them in a computer program. In the next section we will explore the notions of *data type*, *attribute*, *property*, *value*, *object*, and *array*. Together these notions will prove sufficient to construct a system which can encode arbitrary documentary data.

2.6 An object model of documentary data, and a JSON implementation

2.7.1 Precedents

It is important to point out that in general terms the model described here is not a novel structure: the model described below has clear precedents in the literature. The question of how best to go about digitizing documentary data has been well studied: see Zaefferer ([2006](#)), Nordhoff ([2012](#)), to name a couple. These and other researchers have repeatedly recognized what might be called “basic” structures in documentary data. This is a good sign: it means that there is some conventionalized agreement within the field as to what kinds of information a documentary database should contain.

In particular, Bow, Hughes, and Bird ([2003b](#)) *Towards a general model of interlinear text* and Bow, Hughes, and Bird ([2003a](#)) *A four-level model for interlinear text*, have served as touchstones in the development of the model used here. In Bow, Hughes, and Bird ([2003a](#)), the authors develop a logical model of the structure of interlinear text, one which is quite close to the model which will be employed here for corpora. In Bow, Hughes, and Bird ([2003b](#)), the authors carry out a detailed survey of a range of interlinear texts in various formats, analyzing how their structures can be compared and conceptualized as data. In the lexical domain, a variety of work has also been carried out, with many relevant observations dating back decades. Hsu ([1985](#)), is an insightful work, especially given the fact that is a

software manual, describing how dictionary data can be modeled using the productive LEXWARE package. Other relevant works include: Jacobson, Michailovsky, and B. Lowe (2001), Michailovsky et al. (2014), Thieberger (2004), Michaelis et al. (2013), Goodman et al. (2015), and Palmer and Erk (2007) *inter alia*.

The novel aspect of the current work is not, then, the particular details of the data model we will use as the basis for applications. It is the fact that we are striving to represent all kinds of documentary data in a unified fashion, in a single platform, and then build on that structure to develop an approach to creating applications for manipulating such data. This kind of unity in design is a prerequisite for the kind of ubiquitous interlinking that Heath sought in his print database of Nungubuyu.

In order to define a data model which can be effectively represented in a computer, however, we must be explicit about the design and structure of that model.

2.8.1 Machine-readable data types with JSON: attributes, objects, and arrays

In the previous section, we discussed the conceptualization of the notion of a “car”. We suggested that in some contexts, a car might be represented as a year/make/model triple. Each of these three pieces of data might be viewed as an *attribute* which consists of pairs of *properties* (names) and *values*. In our introduction, describing a car “type” consisted of adding a row to a spreadsheet with those properties as “headers”, and then filling in the values in each corresponding cell, as in the leftmost tabulation in the display below. But this is not, of course, the only possible formulation of this data.

The difference between the two tabulations seems trivial: both are mere rearrangements of the same information. In the left tabulation labels are placed above their corresponding values, whereas in the right tabulation labels are placed alongside them:

Figure 2.7 - Two tabulations of the car data

year	make	model
1958	Plymouth	Belvedere

year	1958
make	Plymouth
model	Belvedere

It is worth considering just how conventionalized our interpretations of such tabulations are — it is the typographical differentiation of **properties** as opposed to values that aids in interpreting the relationships between properties and values, even the unfamiliar layout used in the rightmost tabulation.

As discussed in general terms above, the “Car” data type is represented here as a particular set of properties (year/make/model) and with appropriate values for each. In the context of computer programming, such a notion of a bundle of property/value pairs is often referred to as an *object*. In practice and pedagogy of “object-oriented programming”, the notion of objects includes other features, but for our purposes we will focus primarily on this simple, data-oriented representation. This simple notion will serve us well as we study means of recording documentary data in a persistent way, one that can be readily processed by computer programs, displayed

in various formats, and loaded into interactive user interfaces — that is to say, applications.

When a computer program runs, objects are “created in memory”. The details of how this is actually achieved are beyond the scope of this discussion; we may simply conceptualize this process as a means of creating “virtual” objects whose attributes maybe retrieved by property names. There are various functionalities which programming languages impart to each object when they are added to memory. For instance, it is possible to retrieve the values of a particular object’s attributes by referring to their corresponding properties. We may imagine such an exchange as a sort of dialog between a programmer and the computer:

Computer: I have an object which is a car.

Programmer: What is the value of the property called `year`?

Computer: 1958.

Programmer: What is the value of the property called `make`?

Computer: Plymouth.

But of course, computers do not “converse” in natural language, it is through a programming language that such a “conversation” is carried out. We will not delve into that kind of programming here, but we will discuss one notation system which is widely used in many programming languages, known as JSON for Javascript Object Notation. Although its origins are in the Javascript programming language, the notation is widely used as a data exchange format.

One way to “get an object into” a program is to record it in a machine-readable

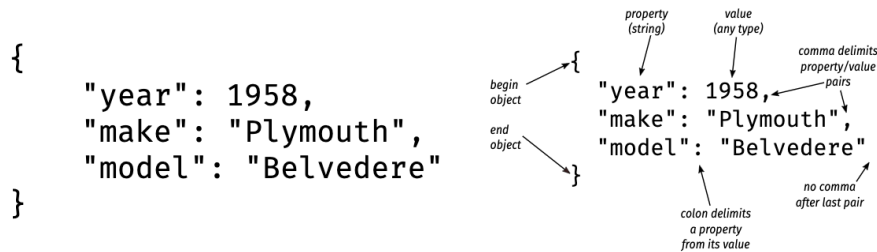
form, which the program can then load into memory. Many modern computer programming languages — including for example languages which prioritize learnability, such as Python, Javascript, and Ruby — have built-in mechanisms for using objects as described above. While the syntax for encoding objects differs slightly from programming language to programming language, a slightly formalized version of the notation which is used by Javascript — hence the name “Javascript Object Notation” — has become a standard across languages, precisely because it is both simple and usable in distinct computing platforms. JSON will serve as the primary data storage format for the applications developed in this work.

Machine-readable formats do not rely on visual characteristics (such as those used above to differentiate properties and values). Rather, they are defined via strict syntactic rules about sequences of characters. JSON is just one such format. HTML and CSS are others. Programs in the Javascript programming language (which together with HTML and CSS is one of the default computer languages of the web platform) are also written as such “plain text” in accordance with a strict (and complicated) set of syntactic rules. The rules for JSON, which has become a de-facto standard data exchange format on the web, are the simplest of any of the four web platform languages.⁷ All JSON data can contain only the following types of data:

Value type	Brief explanation
strings	Textual content. Strings must be contained in "double quotes".
numbers	Numerical values. Unquoted numbers which may be integers (like 5) and decimals (1.5).
true or false	Unquoted special values used in logical operations. These are referred to collectively as "Booleans".
objects	As described above. Each property is delimited from its value by a colon (:), and each such pair is in turn delimited by a comma. The entire object is wrapped in "curly" brackets ({ }). See Figure 2.8 below.
arrays	Lists of any other values, enclosed in square brackets ([]). with each item delimited by a comma (see below). See Figure 2.9 below.
null	An unquoted keyword used to indicate an empty value (not used at the current time in <code>docling.js</code> — empty strings are used instead to simplify rendering.)

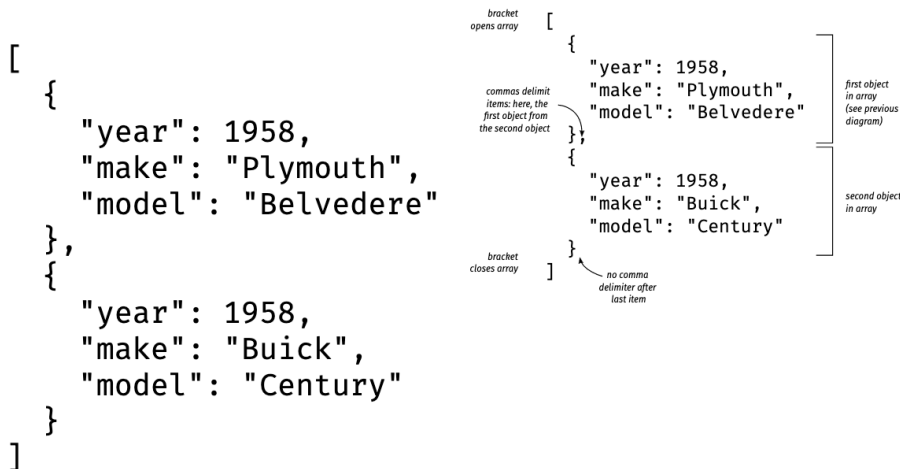
The diagram below demonstrates how the car object we have been discussing could be represented as a JSON object.

Figure 2.8 - JSON representation of an object, with annotated diagram



And the two-row spreadsheet view might be built from data such as the following:

Figure 2.9 - JSON representation of an array of two objects, with annotated diagram



The first represents the car as a single *object*. An object is notated between curly brackets. Within the curly brackets, each property/value pair is separated by a comma. The property label must be surrounded by double quotes (e.g., "year"), but the value may or may not have quotes depending on the specific type of value; here,

we have represented a year as an (unquoted) number, whereas the make and model are quoted and are thus stored as “strings”.

In the second code excerpt, the description of the car as an object appears again exactly as it did in the first excerpt, but in this case it is “nested” within an array. The easiest way to conceptualize an array is to simply note that the rows of the “spreadsheet view” containing two cars is a list with two objects of the same data type: they contain the same properties (such as "model"), while each object’s values for those properties differ ("Belvedere" vs "Century").^{[8](#)}

2.9.1 Documentary data types

Because the data about the Belvedere and the Century have the same properties, and the values corresponding to the properties are of the same kind of information (textual make and model, numerical year), we can say that each is an *instance* of the same *data type*. We can also say that the spreadsheet-style table represents an *array* of such instances. Because the graphical presentation of the spreadsheet format requires that every row share column headers, we can say that the array of instances itself can be thought of as having a data type.

We are now in a position to better understand the data introduced in the small example application in the previous chapter. We can recall that before the user fills out the translations of the suggested prompts, the interface looks like this:

Prompt	Translation
one	<input type="text" value="uno"/>
two	<input type="text" value="dos"/>
three	<input type="text" value="tres"/>

gloss	form
one	uno
two	dos
three	tres

Data

**Figure 2.10 - Translate prompt interface
(fill out translations and press enter)**

Note that the spreadsheet-style tabulation on the right is like an empty spreadsheet: it has column headers, but no content in its rows.

Now the data consists of an array of three “word” objects. In the same way that the year/make/model triple has a special status in identifying a particular kind of car, the form/gloss pair has a special status in identifying a particular word. We will have much more to say about the structure of this data type in Chapter 3.

Before leaving the topic of data types and how they are structured, we need to address the topic of *nesting*. So far, all of the arrays of objects that we have considered and visualized with the spreadsheet-style tabulation have been truly “tabular”. That is to say, our sample data thus far has consisted of rows of data, where each value is “simple” in the sense that it is either textual — a “string” in programming terminology — as in `make`, `model`, `form`, or `gloss`, or a number as in `year`. But while the “features” property for each car listing was represented in our model as a simple string, in fact, we could also conceptualize those values as a list — an array. After all, the string is comma-delimited:

4 door, radio, heater, automatic transmission, 31000 miles, like new

Rather than think of the “features” attribute in this way, we could also think of them as a simple array of strings:

4 door
radio
heater
automatic transmission
31000 miles
like new

```
[  
  "4 door",  
  "radio",  
  "heater",  
  "automatic transmission",  
  "31000 miles",  
  "like new"  
]
```

Figure 2.11 - Visualization of “features” as a simple list of strings, and as JSON

We can take such a simple table and insert it as the *value* of an object property as follows:

year	1958
make	Plymouth
model	Belvedere
features	4 door
	radio
	heater
	automatic transmission
	31000 miles
	like new

```
{  
  "year": 1958,  
  "make": "Plymouth",  
  "model": "Belvedere",  
  "features": [  
    "4 door",  
    "radio",  
    "heater",  
    "automatic transmission",  
    "31000 miles",  
    "like new"  
  ]  
}
```

Figure 2.12 - An “object view” of a single car

Here, the value of the features property is itself a nested list. We could just as

easily represent the same information as an array of objects:

year	1958
make	Plymouth
model	Belvedere
features	feature
	4 door
	radio
	heater
	automatic transmission
	31000 miles
	like new

```
{
  "year": 1958,
  "make": "Plymouth",
  "model": "Belvedere",
  "features": [
    { "feature": "4 door" },
    { "feature": "radio" },
    { "feature": "heater" },
    { "feature": "automatic
      transmission" },
    { "feature": "31000 miles" },
    { "feature": "like new" }
  ]
}
```

Figure 2.13 - Treating the feature list as an array of complex objects

The only visual difference in these two structures is whether each individual “feature” is represented as a plain string or an object. If we wanted to add more elaborate information about each feature (as opposed to the simple one-string-per-feature representation in the previous structure), we could do so here — perhaps adding the feature’s price, etc. Both the array-of-strings and the array-of-objects representations are useful in different contexts.

But enough automotive examples. This will all be clearer when we begin to consider documentary data, and as we explore the design and implementation of applications in Part II. This brief foray has given us enough complexity in data to introduce essentially all of the mechanisms that we need to represent arbitrary documentary data. As we shall see, data in documentary linguistics is filled with nested information of the type described in the “features” in the car database. We now proceed with a brief overview of all the data types necessary for our “Boasian”

database of documentary data.

2.10 Designing a Boasian Database

The “Boasian trilogy” of grammar, corpus, and dictionary is a well-established convention for recording documentary data: it remains highly cited within the language documentation literature ([Woodbury 2011](#); [Chelliah and De Reuse 2010, 14](#); and [Ameka, Dench, and Evans 2006](#), etc). For a useful, brief survey of how texts, grammars, and dictionaries have influenced the development of linguistic theory, see ([Rice 2011, 192–94](#)). As mentioned above, the utility of viewing disparate kinds of documentary data as a unified “trilogy” is that it expresses the way in which that disparate data is interrelated. One might say that the conventionalization of the notion of a trilogy as a model for documentation is a kind of collective acknowledgement that all of those elements must be present for it to be said that a language has been reasonably well documented.

But the Boasian trilogy is typically defined in fairly general terms. What constitutes a dictionary, for instance? Is a simple list of words with brief definitions sufficient, or should the linguist attempt to emulate a 30,000-entry encyclopedic work such as the *Hopi Dictionary of the Third Mesa Dialect* with head words, combining forms, form classes, usage examples, definitions, morphology, loanwords, and cross-references ([Hopi Dictionary Project 1998, xvii–xviii](#))? Different languages require different information by their very natures, different researchers will focus on different aspects of the language, and so forth. Just what constitutes a “text” or

“grammar” are equally complex issues.

Yet, the frustration mentioned in the previous chapter seems to arise in part from a disconnect between the way software presents documentary data and linguists’ own conceptions of what the software “should” do: if linguists are frustrated that their tools do not allow them to “manipulate and arrange” their data as they see fit, what exactly does that *mean*, in terms of the actual information stored and the actual processes that the linguists wish to carry out with user interfaces? Clearly, there is a received if diffuse understanding of what variety of information the Boasian trilogy contains, but how should a “Boasian database” actually be designed? And how can we capture the notion that all three parts of the trilogy should be explicitly linked to each other? The remainder of this chapter describes one answer to this question.

2.11.1 A catalog of data types in docling

Language Data Type

The **Language** object contains basic metadata about a language, information about the phonetic inventory, and orthography. It is useful to compile this information into a single object, as information such as orthographic transliteration may need to be referred to from many points within an application. Much more could be said about the structure of the **Language** object, but the following sample is sufficient to give some idea of the key objects that can be linked together.

metadata	name	Rapa Nui			
	family	Austronesian			
	source	Du Feu (2012)			
orthography	ipa rapanui				
	k	k			
	ʔ	ʔ			
	m	m			
	n	n			
	ŋ	g			
	p	p			
	r	r			
	t	t			
	β	v			
	a	a			
	e	e			
	i	i			
	o	o			
	u	u			
inventory	vowels	letter	height	backness	rounding
		a	open	front	unrounded
		e	close-mid	front	unrounded
		i	close	front	unrounded
		o	close-mid	back	rounded
		u	close	back	rounded
	consonants	letter	voicing	place	manner
		k	voiceless	velar	plosive
		ʔ	voiceless	glottal	plosive
		m	voiced	bilabial	nasal
		n	voiced	alveolar	nasal
		ŋ	voiced	velar	nasal
		p	voiceless	bilabial	plosive
		r	voiced	alveolar	tap
		t	voiceless	alveolar	plosive
		β	voiced	bilabial	fricative

Rapanui Language

Rapanui Language

Note that this is only example data. The object which is the value of the `metadata` property is arbitrarily extensible, as necessary. For instance, we could elaborate the metadata with language codes, geographical region, and family subgrouping information:

metadata	name	Rapa Nui	
	family	Austronesian > Malayo-Polynesian > Oceanic > Polynesian > Eastern Polynesian	
	source	Du Feu (2012)	
	geographicalRegion	Easter Island	
	codes	glottocode	rapa1244
		iso-639-3	rap
orthography	...		
inventory	...		
Rapanui Language (more extensive metadata)			

(Such extensibility should be borne in mind throughout the following examples; in each data type's metadata value, the fields included are only intended to give an idea of what kind of information a linguist might find useful — metadata might be much more elaborate in a given project.)

Word Data Type

As mentioned above, the **Word** data type is defined quite simply: it is an object with at least *form* and *gloss* attributes:

attribute name	description
form	morphological transcription
gloss	morphological analysis

Word Data Type

Examples in this section are from the author’s fieldwork on Hiligaynon:

Leipzig glossing notation may be used (in both values) for morphologically complex words:

form	subóng
gloss	now

Sample
Hiligaynon
Word

While the *form* and *gloss* attributes suffice to *identify* a word, they most certainly do not *exhaust* the kinds of information with which words may be annotated. The full set of annotations which apply to each word is stored in a single place — a dictionary object or “entry” object — rather than every time the word occurs. As we shall see, even the tokens of words (within syntactic contexts, rather than dictionary entries), may be annotated with arbitrary information as the linguist sees fit. One might choose to add an *orthographic* field and a “phrasal” translation, for instance:

orthographic	nagapungko
form	naga-pungko
gloss	CONT-sit
phrasal	is sitting

Sample Hiligaynon Word

Similarly, a word from a tone language such as San Martín Duraznos Mixtec might be considerably elaborated, including additional fields indicating tonal information. One approach to handling multilingual values is also presented here, with the fields `tone` and `gloss` also made available in a Spanish localization.

form	ˢtaʔaɭ	
gloss	hand	
orthographic	nta'à	
tone	ML	
tones	es	MB
	en	ML
glosses	en	hand
	es	mano

Sample San Martín Duraznos Mixtec Word

All such decisions are intended to be extensible; we will discuss the design and use of **Word** objects more fully in Chapter 3.

Sentence Data Type

The choice of the term “sentence” to represent the “unit of interlinear text” is one of the most problematic in this system, as opposed to other labels such as “phrase”, “intonation unit”, “utterance”, and so forth. However, like all of the labels for the objects in this catalog, we are speaking only of a convention for data representation.

For this reason the most general, generic term has been chosen whenever possible. It should also be noted that these terms will only explicitly be used within data files and “behind the scenes” of user interfaces. The following example in Hiligaynon demonstrates minimal attributes for a **Sentence** object:

attribute name	description
transcription	phonemic transcription
translation	free translation
words	array of Word objects corresponding to the transcription

Sentence data type

Note that the nested **Word** objects have the same relationship to the **Sentence** object that the “features” had to each car listing in the non-linguistic example above. Note that the fact that the **Words** have internal structure (i.e., both *form* and *gloss* attributes) necessitates their representation as objects rather than as strings.

transcription	Baw indí ko guid malipatán ang mga memories sa CPU.	
translation	Wow, I really can't forget my memories at CPU.	
words	form	gloss
	baw	so
	indí	NEG
	ko	1S.ERG1
	guid	very
	ma-lipat-án	IRR.COMPL-forget-LOC
	ang	FOCUS
	mga	PL
	memories	memories
	sa	to
	CPU	CPU

Sample Hiligaynon Sentence object

We will see some modifications of this basic structure in Chapters 3 and 4.

Text Data Type

The **Text** object primarily functions as a container for an array of **Sentences**. As we saw above for the **Language** object, the **Metadata** may be elaborated with additional fields as necessary. We will have much more to say about what kinds of metadata one might want to insert into documentation at many levels of analysis in later chapters. (Metadata need not be limited to the **Text** level by any means.)

The Text object contains three levels of nesting. Word objects are nested as arrays, and those arrays are in turn nested in Sentence objects, and finally, an array of Sentence objects are nested in Text objects.

Designing a Boasian Database

metadata	language	Hiligaynon			
	source	https://www.youtube.com/watch?v=cUqMWG4QJmK			
	speaker	Juan Lee			
	linguists	Joshua De Leon			
		Patrick Hall			
sentences	transcription	translation	words		
	dirá ko na kwa' akón mga knowlédge	That's where I got all my knowledge.	form	gloss	
			dirá	there	
			ko	1S.ERG1	
			na	link	
			kwa'a	get	
			akón	1S.ERG1	
			mga	PL	
	knowlédge	knowledge			
	nu?	know?	form	gloss	
		nu	know		
sa mga maéstra ko si CPU	From my teachers at CPU	form	gloss		
		sa	to		
		mga	PL		
		maéstra	teacher(F)		
		ko	1S.ERG1		
		si	PERS		
		CPU	CPU		

Sample Hiligaynon Text extract

Sample Hiligaynon Text extract

Corpus Data Type

Just as **Text** objects contain an array of **Sentence** objects, a **Corpus** object contains an array of **Text** objects. At this level of nesting, our tabulation visualization — thus far useful for condensing information in a compact, generic form — starts to become difficult to read easily. For this reason, the tabulation below

demonstrates how texts are nested in a corpus object, but leaves out the content of the texts, and has only cursory metadata for each.

metadata	title Small Hiligaynon corpus	
links	type	file
	text	Education in Jaro data/languages/hiligaynon/corpus/education_in_jaro/education_in_jaro-text.json

Sample small Hiligaynon Corpus

Note that the corpus object does not itself contain an array of `Text` objects itself — rather, it contains an array of *references* to `Text` objects. This is a much more manageable approach to storing text data, for several reasons. For one, there should be no *a priori* constraints on how `Text`s are grouped into corpora. Indeed, a single text may profitably be included in multiple corpora: obviously, the sample Hiligaynon text included here is part of the corpus produced in the particular fieldwork class in which it was created. But it is easy to imagine other hypothetical corpora: a corpus of Hiligaynon monologues collected from the internet; a collection of first-person accounts of education in the Philippines, and so forth.

Lexicon Data Type

I am not indulging in trivialities when I point out that even if all the meanings of the words are wrong, the dictionaries maintain their value as word collections.

Benno Landsberger, quoted in Oppenheim ([1968](#))

A brief aside with regard to the relationships between corpora and dictionaries is relevant here. There is a paradox in the history of the Boasian Trilogy ([Chelliah and De Reuse 2010, 227](#)). Namely, while the original advocates of the notion of the Boasian Trilogy (including Boas himself) produced many grammars and corpora, they produced few if any dictionaries. As Chelliah points out, Boas himself never published a dictionary, nor did Bloomfield; Sapir published just one, on Southern Paiute [Chelliah and De Reuse ([2010](#)) p. 227]⁹. The lack of published dictionaries is easily ascribed to the tedium of producing them, given that it can take far longer than producing a grammar or corpus. And yet, if we step back and consider the difference between *identifying* words and *defining* words, the paradox becomes less mysterious.

In a (modern) interlinear glossed text, each word is accompanied by a morphological gloss. Indeed, most linguistics journals and publishers require that morphological glosses be included in accordance with the Leipzig Glossing Rules ([Comrie, Haspelmath, and Bickel 2015](#)) and the Generic Style Rules for Linguistics ([Haspelmath 2014](#)). While a morphological gloss is in no sense as nuanced as a prose dictionary entry with a prose definition, multiple senses and other apparatus, it does serve a crucial function in documentation: *together* with the form that it glosses, a gloss can serve to uniquely identify a word. If the full set of words in the interlinearization of all the sentences in a corpus are aggregated, duplicates removed, and sorted, the result is a basic but useful representation of the known word-forms in a language.

Viewed from this point of view, while it is true that the “trilogists” did not in fact

create traditional dictionaries, they did identify all the words in their corpus uniquely, via a recording of the word's phonetic form and some unique representation of the word's meaning and structure. Here, we will refer to such a list of unique words which is identified by a form and a gloss as a *lexicon*. Like many of the terms we will use in this work, the term "lexicon" can mean many things in many different contexts in linguistics (the "mental lexicon", "lexicography", etc.). We do not dispute those usages, we are simply specifying a label for this sense of "a collection of unique glossed forms".

Perhaps the simplest way to conceptualize the notion of **Lexicon** as it will be used here is to imagine that all of the **Words** which occur in a complete corpus are removed from their sentential contexts, sorted into a single array, and then duplicate *form/gloss* pairs removed. This list of unique **Words** is a **Lexicon**.

A personal aside on "data guilt"

Fieldwork data is messy. It is essentially never as finished as we would like it to be. Every element in documentation is subject to change, revision, or even removal. Of course, this state of affairs is in direct conflict with academic ideals for writing and publishing, where we strive to make our work error-free and entirely defensible.

How are we to resolve this conflict? In the opinion of this author, the answer is that we fight back against our own inclination to perfect every jot and tittle of our documentary work before we share it with our colleagues. To borrow a term from the description of verbal

aspect, we should think of the production of a documentary database as an atelic event, not a telic one.

And it is for this reason that I am sharing my own incomplete documentation: some of the annotations in example data structures in this chapter are confused, incomplete, or simply wrong. Looking back at my work, why did I gloss both **ákon** and **ko** as 1S.ERG1? Quite frankly, I don't remember. I can see that those glosses need to be fixed. In fact, my entire analysis of voice marking in pronouns needs revision. It is quite difficult for me to resist the urge to correct these errors before including them in this document — I have strong feelings (guilt, embarrassment, and worse) linked to my own documentary analysis. But if we are going to tackle the problem of interacting with documentary data in a holistic manner, we must also face the simple fact that our databases are and will always remain imperfect. The more quickly we transfer our data into a familiar, legible format, the more likely we are to notice errors.

Let us consider all of the words from the three-sentence **Text** example above. Note that this is from *incomplete fieldwork*.

form	gloss
dirá	there
ko	1S.ERG1
na	link
kwa'a	get
akón	1S.ERG1
mga	PL
knowlédge	knowledge
nu	know
sa	to
mga	PL
maéstra	teacher(F)
ko	1S.ERG1
si	PERS
CPU	CPU

These words are in their order of occurrence in the three sentences in the sample text. If we sort them by form, we see some repetitions:

form	gloss
CPU	CPU
akón	1S.ERG1
dirá	there
knowlédge	knowledge
ko	1S.ERG1
ko	1S.ERG1
kwa'a	get
maéstra	teacher(F)
mga	PL
mga	PL
na	link
nu	know
sa	to
si	PERS

The word **mga** ‘PL’, the Hiligaynon plural marker, occurs twice in these sentences, as does **ko** ‘1S.ERG1’. For the purposes of a **Lexicon**, we wish to remove such repetition. Having done so, the array looks like this:

form	gloss
CPU	CPU
akón	1S.ERG1
dirá	there
knowlédge	knowledge
ko	1S.ERG1
kwa'a	get
maéstra	teacher(F)
mga	PL
na	link
nu	know
sa	to
si	PERS

The distinction is minimal in this short list, but it balloons quickly as corpora grow. (The difference between the full list of tokens in a corpus as compared to the full list of *unique* words in a corpus varies in accord with Zipf's Law ([Manning, Raghavan, and Schutze 2008, 1:82](#)).)

In order to improve the portability of an instance of the **Lexicon** data type, we will also include a *metadata* attribute with key information, and the list of words will be placed under a *words* attribute, as follows:

attribute name	description
metadata	basic descriptive metadata about the lexicon
words	array of unique words

Lexicon data type

And here, then, is our small in-progress working lexicon of Hiligaynon, such as it is:

metadata	title	Hiligaynon working lexicon
	language	Hiligaynon
	compiler	Patrick Hall
	notes	Based on data originally gathered during 2014 UCSB Fieldmethods Class. Instructor: Marianne Mithun; Students: Di Caminsky, Patrick Hall, Elliott M. Hoey, Megan Lukaniec, Heather Simpson. Mistakes are this author's.
words	form	gloss
	CPU	CPU
	akón	1S.ERG1
	dirá	there
	knowlédge	knowledge
	ko	1S.ERG1
	kwa'a	get
	maéstra	teacher(F)
	mga	PL
	na	link
	nu	know
	sa	to
	si	PERS

Grammar Data Type

The **Corpus** data type is probably familiar in broad outline to documentary linguists. The notion of the **Lexicon** as a list of unique *form/gloss* pairings (as opposed to dictionary entries), is perhaps somewhat less familiar as a “format”, but it is easily understood as a highly constrained kind of word list. The way that we will define the **Grammar** here, however, is probably quite unfamiliar. A full development of this data structure and its application in `docling.js` applications is beyond the scope of the current work, but here is a simple example of some key fields:

attribute name	description
category	the category to which a linguistic element belongs
value	the particular value of that category
symbol	a unique identifier for the category/value pair

Grammar data type

Here is a sample grammar table for a language whose grammatical categories may be familiar to many readers, Latin:

Describing such a structure as a “Grammar” may seem, at best, presumptuous. How can the most complex part of documentation — the grammar — be reduced to nothing but a list of categories? It is better to think of this list as a set of symbols which point to particular configurations of grammatical features; symbols such as these are used throughout the data model used in `docling.js`. The Grammar data

object serves as a single source of truth for the denotation and symbolization (via abbreviations) of grammatical category labels in the database. The terms in this particular list are derived from a particular source, Moreland and Fleischer (1990). (Note the use of mixed-case abbreviations and periods, no longer current practice). We shall see that many other kinds of grammatical categorization, however, may be expressed in the same way — including categories which are not familiar from Leipzig-style morphemic glossing abbreviations. The locus for application of these categories is not constrained to the morphological context; we may use the same notion of category/value pairs to create useful labels for categorizing elements at any level of Boasian database, and for constructions which may have complex grammatical structures. Such labels may refer to analytical levels — a label for a voice construction, for instance, may make reference to linguistic units at the morphemic or syntactic levels, for instance.

2.12 Modeling data change with dataflows

At this point we may proceed to define the notion of a “dataflow”, introduced in the previous chapter, in a more detailed fashion. Recall that in that chapter, a dataflow was defined as the set of changes which convert one state of documentary data into another state. Thus, in the sample lexical elicitation application (§1.3.3.1) the dataflow consisted of adding values for the *form* attribute for each word in a list.

But that is a very simple example, involving only the modification of a single attribute of a particular data type (the *form* of a **Word** data type). The notion of a

dataflow is intended to be very generic: a dataflow may involve any transition from one state of data to any other. Initially, we will constrain the kinds of modifications we make to operations involving instances of the data types cataloged above, but it is of course possible to extend that list.

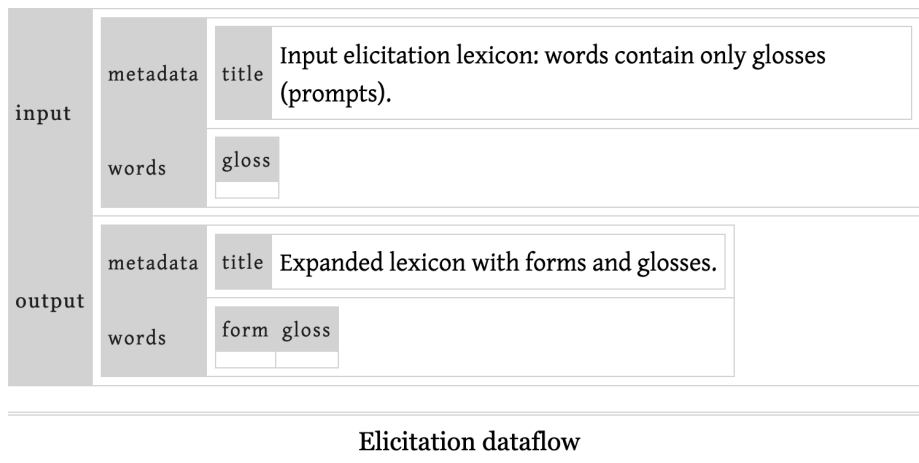
2.13.1 Familiar documentary dataflows

Note that the dataflow itself is not a definition of a user interface “UI” or what is sometimes called “user experience” or “UX”. What it does define is a logical representation of input and output states of some abstract documentary task. That logical representation, however, helps to better define some of the working terminology documentary linguists make use of as they describe the kinds of work they are carrying out. Indeed, dataflows are a convenient way to formalize the meanings of some of these familiar terms.

To read the dataflow diagrams, compare the structure of the input with that of the output. The difference between the two is the substance of the dataflow. Thus, in the elicitation dataflow, forms are added; in glossing, glosses are added; in tokenization, an array of word forms is added; in time-alignment, links to media are added, and so forth.

Elicitation

For example, the elicitation dataflow viewed above may be defined in a manner that is only slightly more verbose than the presentation in Chapter 1:

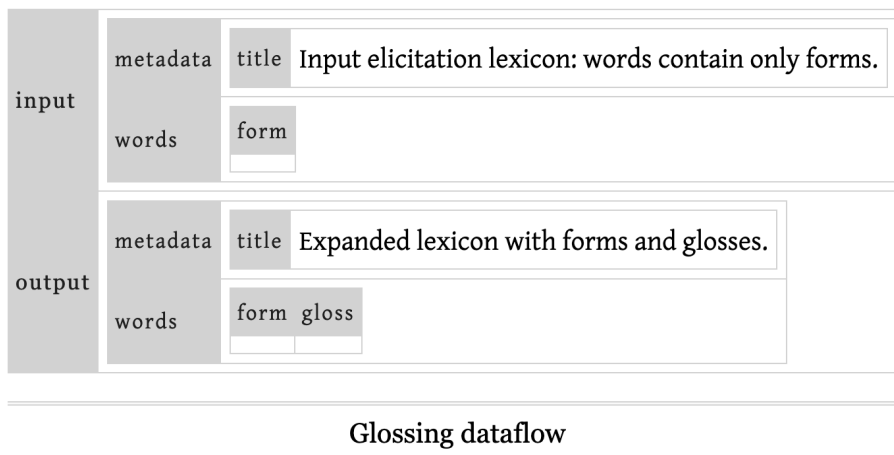


Here, we have explicitly indicated the input and output stages of the dataflow. Note that the values of the *input* and *output* attributes are themselves simply instances of the **Lexicon** data type: the only difference between the information in the two data types is that the output **Lexicon**'s *words* attribute contains **Words** with a specified *form* attribute.

Here are a few more examples of dataflows defined for familiar terms from documentation.

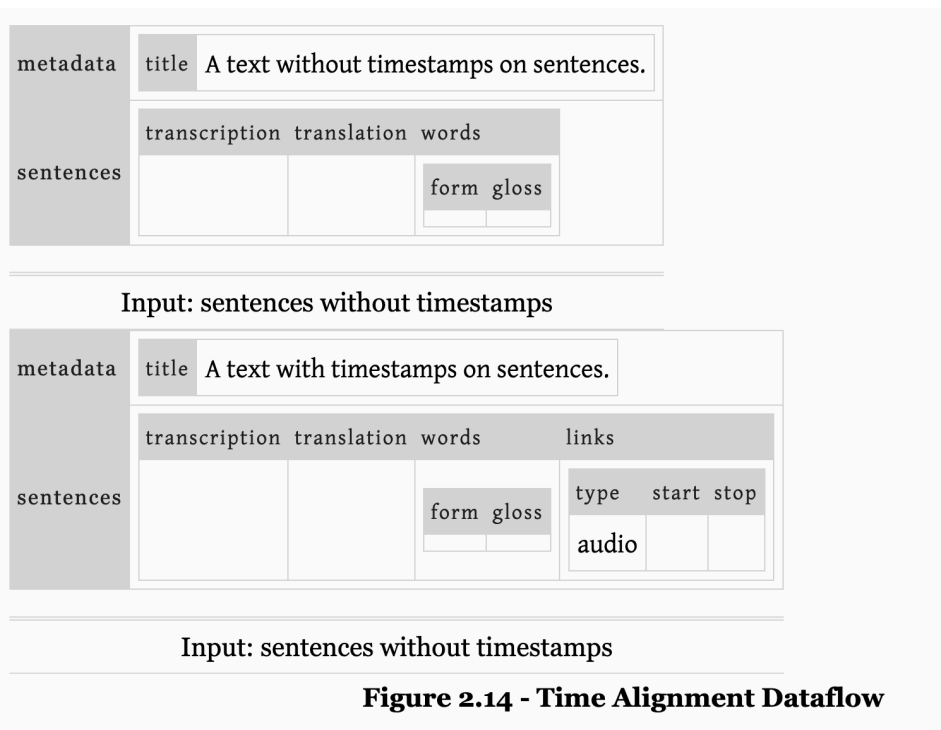
Glossing

Glossing is, in a sense, the inverse of the Elicitation dataflow just presented. In this case, the input is an array of **Words** containing only *form* attributes, and the output is an array of **Words** containing both *form* and *gloss* attributes.



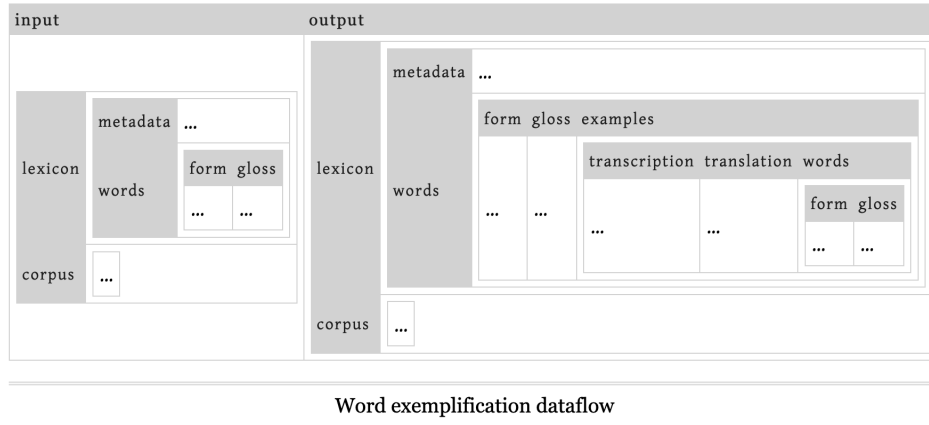
Time-alignment

The familiar operation of adding start and stop timestamps to a sentence (a primary annotative functionality of the ELAN software package) is called “time-alignment”.



Note that in this case media links¹⁰ were added at the level of the sentence; the same type of *link* attribute could be added at the word or even phoneme level. We shall explore these structures in more detail in Chapter 3.

Word Exemplification



```
[ { "input": { "lexicon": { "metadata": "...",
  "words": [ { "form": "...", "gloss": "..." } ] },
  "corpus": [ "..." ] }, "output": { "lexicon": {
  "metadata": "...", "words": [ { "form": "...",
  "gloss": "...", "examples": [ {
  "transcription": "...", "translation": "...",
  "words": [ { "form": "...", "gloss": "..." } ] } ] },
  "corpus": [ "..." ] } } ]
```

In this dataflow, an array of “example” sentences is added to each word in a

lexicon. The examples of each word are drawn from the corpus. This output of this dataflow is comparable to a typical use of the slipfile as described by Munro in Chapter 1. This output is essentially identical to Heath’s dictionary to corpus reference system as described at the beginning of this chapter, where the dictionary entry for the word *dhan^sid!* was followed by a series of cross-references to a corpus. Note that in the dataflow shown above — and in this regard the data structure is unlike Heath’s “print database” — the example sentences are not simply *referred* to within the entry data, rather, full copies of the sentence data structures are “embedded” within the word data structure itself.

The notion of embedding full example sentence data within the word entries as described here raises a number of questions of efficiency as far as implementation is concerned. Crucially, it casts the example sentences themselves as being encoded redundantly — once in the corpus, and once in the dictionary. It is important to bear in mind that in actual implementation, “behind the scenes” mechanisms provided by programming languages and database tools are designed to handle such issues. In fact, there are many, diverse methods of doing so. The details of how this works are beyond the scope of the current work, but suffice it to say that an example sentence which appears in a dictionary display in one interface and a corpus in another may in fact be stored just once. Here, we are concerned with presenting an interpretable data model of documentary data that is familiar and understandable to people — to working linguists. Because linguists are familiar with the notion that example sentences within dictionary entries are “drawn from” corpora, we simply stipulate that such data linkages must be supported and leave the particular implementation to a later discussion of the source code itself.

We shall see in Chapter 5 how this dataflow output may be used to “render” an variety of familiar notational formats, especially those familiar from dictionary entries with example sentences. We shall also see that exemplification of this kind may be generated automatically.

Because we have described these examples only in terms of arrangements of data types, they may seem at this point to be rather too abstract. More examples will be developed and worked through with sample data in Chapters 3 and 4, but it is hoped that this collection of dataflow descriptions provides, at least, some evidence that the data model we have described is useful. These precise definitions of dataflows will be helpful in Chapter 4, in which we address the problem of defining actual steps — which we will refer to as workflows — which, when completed by the linguist, result in the fulfillment of the dataflow operation.

As we have seen, orienting ourselves towards “objects” in documentary data provides a useful way of characterizing both the nature of documentary data (data types), and processes by which it may be altered (dataflows). This viewpoint helps to shed some light on the precise nature of the “sorting plague” that Bloomfield and Harrington lamented as so laborious and time-consuming. Because their materials were physical — notebooks, slipfiles, and other paper artefacts — physically rearranging those materials in order to “carry out” a data flow such as word exemplification would have been a daunting prospect. And while details of how such “collation” (to borrow Bloomfield’s label) was carried out are scant in the literature, it is clear that sorting a database of thousands upon thousands of physical objects would have been excruciating. Considering also that a single sentence may serve as

an exemplification of any of its constituent words, it seems likely that fileslips would have been produced for every word in a sentence, in order that each fileslip could be found exemplifying each of those words — resulting in massive duplication of data. Compared to such Sisyphean efforts, the power of a mid-range laptop is astonishing. All data in a computer, after all, can be treated as a virtual entity, and duplicated and rearranged almost at will, and in the blink of an eye.

In the next chapter, we will turn from the structure of documentary data to approaches to designing and implementing software which supports that data structure.

3. Viewing Documentary Data

In the previous chapter we considered an abstract model of documentary data which can be encoded as JSON objects and arrays. That model is useful for thinking about “pure” documentary data, and for storing such data in a reliably simple way, but of course a data format such as JSON is not itself acceptable as a user interface: we do *not* want to directly edit JSON files. Rather, we need effective ways to *display* that data legibly. We will build such a system “on top” of HTML itself, in the form of *web components*: “custom” HTML elements which are specifically designed to 1) display documentary data and 2) make such data interactive and editable.

We [saw in Chapter 1](#) how the HTML markup language can be used to encode structured documents. HTML is composed of a set of default “tags” which serve various purposes within an HTML document. Some are used to break text into paragraphs (the `<p>` tag). Others indicate that a run of text is important (``, typically rendered with a **bold** typeface) or emphasized (``, typically *italic*). The logical hierarchy of different sections of the document may be set off with headings and subheadings of various levels (`<h1>` being the highest, followed by `<h2>` down to `<h6>`) — the section you are currently reading begins with an `<h2>` heading, while the entire chapter begins with an `<h1>`. For encoding data which are in a tabular relationship, there are a whole series of tags, including `<table>`, `<tr>` for rows, `<td>` for cells, and several more.

But there are also many default HTML tags which are unrelated to the conventions for marking up a structured document. Such elements allow the reader to *edit*

content. The `<button>` tag, for instance, makes its content “clickable”: . For inputting text, the `<input>` tag is available: . Other tags include the `<select>` or so-called “dropdown”: . If you tried manipulating these example elements, you might be left wondering why they were included in the page, since they seem to do nothing. But this reveals an important fact about such “interactive” elements: unless functionality is programmed to accompany their presence, interactive elements do nothing. It is up to web developers to define how the basic functionality of such elements should be associated with computations of some kind.

The HTML standard “HTML Living Standard” ([2021](#)) only specifies what the building blocks are — the task of defining how to combine elements into a usable, useful application is what “web development” really involves. It does not answer, in other words, questions such as “what sort of information should be collected in `<input>` tags?”, or “what interactive behavior should occur when the user clicks a `<button>` or makes a selection in a `<select>`?”. In our case, the kinds of information we seek to use is documentary data of the sort described in Chapter 2. The purpose of the `docling.js` library is to bridge this gap between the programmable environment of the web browser, and digital documentary data itself. The library attempts to provide an answer to the question “How can we use the design palette provided by the web-browser programming ecosystem to compose applications which represent the full gamut of data types that are important to documentary linguists?”

Here, then, we begin the process of working through basic user interfaces for all

of the data types described in Chapter 2. The `docling.js` library is responsible for providing coordination between structured documents (HTML), documentary data (which we encode and save as JSON files), and what is sometimes called “user behavior”. User behavior is everything a human being does to interact with an application: typing on the keyboard, controlling the cursor with a mouse or a trackpad, and so forth. In the current work, we will be thinking of web development as more of a *design* problem than as a *programming* problem. The goal is to help the reader build up a mental catalog of building blocks that can be composed into designs for applications that can be implemented. While this book is not a tutorial in programming Javascript — simply reading it will not be sufficient to equip one to *implement* working applications — it does include high-level introductions to three topics which are arguably “behind the scenes” of a working web application. These topics are:

1. How HTML elements work together in `docling.js` applications
2. How CSS (see below) can be used to modify the presentation of those applications
3. How the JSON syntax may be used both to conceptualize abstract data and to persist “physical” copies of documentary data as files

To a significant extent, the `docling.js` library is designed to abstract the details of how these technologies are actually being used behind the scenes. However, to the degree that details are introduced here, it is hoped that linguists will find themselves in a position to evaluate how the web platform can be used to achieve their goals. One might question whether it is reasonable to describe a

software library without delving into the details of the software code itself, but familiarizing oneself with the design of applications helps users to understand what kinds of challenges arise when implementing user interfaces for carrying out documentary tasks. While it is hoped that a subset of readers will be inspired to investigate the code itself, a more critical first step for the field is to empower linguists to understand how a functioning system — the `docling.js` library — can open many avenues in documentation work.

It is the application design process, then, applied to the domain of documentary linguistics, that is the topic of the remainder of this work. In the remaining chapters, we will work through the implementation of user interfaces for displaying and interacting with data of the data types described in thus far (Word, Sentence, Text, etc).

Source code

For interested readers, the source code for every `docling.js` component is available with a test page in the `docling/` subdirectory of the directory in which you are reading this web document:

[docling/](#)

Note that this source code will be soon be available independently at:

<https://github.com/docling-forum/docling.js/>

This link is also available in the chapter navigation bar at the top of this page.

For instance, a test page and brief documentation for the `<text-view>` element is available at:

<docling/text/text-view/text-view.html>

Sample data for this chapter

Except where otherwise noted, the sample data in this chapter is from the author’s unpublished fieldwork on a text in Hiligaynon (Central Visayan, Austronesian), from a university fieldmethods class context. The origin the text is a [YouTube video](#) from a YouTuber named Juan Lee (“Ilonggo Boy”). The author and the speaker from the fieldmethods class, Joshua De Leon, carried out a series of rehearsing sessions to transcribe the Hiligaynon, the (highly imperfect) morphological

analysis is by the author. Interestingly, Lee, the YouTuber, attended Central Philippine University (CPU) in Iloilo City on the island of Panay. CPU provides schooling for all ages, and Lee states that his entire education took place at the same institution “from nursery school” all the way through college. De Leon had attended CPU for high school, and thus was quite familiar with Lee’s observations about life on the campus. Their shared educational experience also makes the text and its rehearsals interesting as a point of comparison between Lee’s use of Hiligaynon and De Leon’s. In terms of data, the text contains some 900 words, and is segmented into 143 sentence units. This is unpublished fieldwork, and it presents all the complexities and imperfections of a work in progress. Nevertheless, the *data structure* — the form in which the transcriptions and analysis are recorded, however imperfectly — has been adapted to the `docling.js` data model. A selection of sample data drawn from a wide variety of languages is also included in this work.

The full repository of sample data is available alongside the `docling.js` Javascript library mentioned above, in its own `data/` directory. For example, the JSON representation of the Hiligaynon text just mentioned is available at:

[data/languages/hiligaynon/corpus/education_in_jaro/education_in_jaro-text.json](#)

3.1 Reviewing key concepts

We have introduced a fairly large list of terminology so far, so we will pause briefly to summarize and distinguish key terms. We will discuss the three-way classification of web components in the next section, but also summarize them here for convenience.

Object

An object is one of two abstract data structures which are used in JSON data.

Data type

A data type is an object with pre-determined property names and particular types of values. For example, a `Word` data type is defined as having properties `form` and `gloss`, each of which has a textual (or “string”) value.

Dataflow

An abstract description of a change in data from one state to another. For instance, we may describe the process sometimes referred to as *glossing* or *defining* as the process of adding a `gloss` property to an object which already contains a `form`: `{"form": "casa"}` becomes `{"form": "casa", "gloss": "house"}`. Note that dataflows are intentionally “agnostic” as to *how* the change is carried out through a user interface.

View components

Views are legible presentations of a data object at *one point* in a dataflow.

Editors

(See Chapter 4) User interfaces which assist the linguist in actually “carrying out” a data flow — either adding or modifying the values of a given object.

3.2 A first look at web components

In [Chapter 1](#), we introduced some of the standard HTML elements. In general, HTML files consist of a series of *tags* which are interpreted by the browser when it loads an HTML page. Because tags may contain not only text but also other tags — that is to say, tags may be nested — a hierarchical structure is encoded into the HTML document, and the browser can refer to both individual tags as well as “sub-trees” within the document’s full tree of nested elements.^{[11](#)} The HTML file is said to be the *source* of what the user sees in the web browser. In the display below, a very simple

HTML file appears on the left, and a simulation of how it would be rendered in a browser appears on the right. In this case we are considering a simple user interface for presenting a “talking dictionary”-style dictionary entry.



With some comparison, the relationship between the “rendered” content and the raw HTML tags on the left should be fairly apparent. The `img` tag, for instance, corresponds to the image of a window. Note that it contains a *src attribute* which is set to the value `window.jpg`, which is “pointing to” a JPG image file. But this example is just standard HTML. Each bit of the rendered display has been hand-written into the HTML: the form **bentana** was inserted into a `` tag, for instance (which, by default, displays its contents in a bold font). An `<audio>` tag with an explicit reference to an `.mp3` file was also included directly, and so forth for the other annotations associated with this word. The display is usable (if not particularly aesthetically pleasing), but hand-coding such content directly as HTML has many drawbacks: it’s very tedious to write so much HTML markup, and doing so

is particularly error prone. Perhaps worst of all, once that markup has been hand-written, any change in the design of the markup (say, for instance, when an author chooses to use some tag other than `` to contain forms) will require further hand-coding of that change in order to propagate it across all existing markup. Clearly, this is not a scalable solution.

It is precisely because of problems such as these that the combination of “bare-bones” data in JSON format and web components is so powerful. Web components are defined programmatically (as Javascript code), but one need not be a Javascript programmer in order to make use of them — one only needs to understand how the web component’s *tag* and its associated *HTML attributes*^{[12](#)} are defined, and how to point it to the data it needs to render. This is because we can design web components in such a way that they can “consume” JSON data (such as a `Word` object), and then insert all the various values from that data into particular empty tags within a kind of “skeleton” HTML template. This is done automatically as far as an HTML author is concerned; the author need only know how to import the web component and specify where it should find its data files. Typically, those data files are JSON files, and we define the web component to expect an HTML attribute called `src` to point to that data file.

Let us imagine a recasting of the previous word entry display as a web component rather than hand-coded markup. We’ll call our custom element `<entry-view>`, and we’ll define it with an attribute called `src` which can name a JSON file.

Let us consider a JSON object which represents the information about a word

displayed in the hand-coded HTML above:

<pre>{ "form": "bentana", "gloss": "window", "metadata": { "audio": "bentana.mp3", "image": "window.jpg", "etymology": { "language": "Spanish", "form": "ventana", "gloss": "window" } } }</pre>				
form	bentana			
gloss	window			
metadata	audio	bentana.mp3		
	image	window.jpg		
	etymology	language	Spanish	
		form	ventana	
			window	

Note that we have added some extra information here, including a reference to an image, audio, and a reference to the word's etymon. In the demonstration below, notice how the HTML markup has been reduced to *two* tags: the first (`<entry-view>`) informing the browser that we wish to use that custom element, and secondly, a `<script>` tag which “imports” the definition of the custom element itself.¹³ Note also that the `<entry-view>` tag is “pointing to” a file containing the JSON data above. Again, in practice we would probably not approach this using a file containing a single word, but rather, a whole lexicon or other data structure which is likely to be useful in description and documentation. This encapsulation is in fact of great benefit, because a simple tag with just a few customizable attributes is very easy to understand and use.

```

<!doctype html>
<html>
<head>
<title>Bentana</title>
<meta charset=utf-8>
</head>
<body>

<entry-view src=bentana.json></entry-view>

<script type=module src=EntryView.js>
</script>
</body>
</html>

```

bentana



Hiligaynon **bentana** *window*

Borrowed from Spanish *ventana*.

The *web component* standard is simply a means of allowing new HTML elements to be created which work in a similar manner. The `docling.js` library is intentionally designed to be specific to the context of web-based language documentation. For example, below we see a demonstration of the custom `<text-view>` component which renders a standard interlinear presentation of a Text object.

```

<!doctype html>
<html>
<head>
<title>Education in Jaro</title>
<meta charset=utf-8>
</head>
<body>

<text-view src=education_in_jaro_sample-text.json></text-view>

<script type=module src=docling.js></script>

<link rel=stylesheet href=docling.css />

</body>
</html>

```

► Education in Jaro

► Search

0:00 / 4:35

8,580-9.650 **Hello, akó si Juan Lee.**

hello	akó	si	Juan	Lee
hello	1S.ABS	PERS	Juan	Lee

Hello, I'm Juan Lee.

9.704-10.714 **matopíc na akó subóng**

ma-topic	na	akó	subóng
IRR-topic	already	1S.ABS	now

I'll start the topic now

10.753-11.818 **parte sa mga eskwélahan.**

parte	sa	mga	eskwélahan
part	to	PL	school

about the schools

The `<text-view>` tag is one component defined in the `docling.js` library. Again, we see the `src` (for “source”) attribute specifying the name of a JSON file (in this case, `education_in_jaro_sample-text.json`). For simplicity’s sake, we are looking at an abbreviated version of the *Education in Jaro* Hiligaynon text with just three sentences. The `<script>` tag imports the `docling.js` library, which includes a component which defines the `<text-view>` tag. Additionally, because the formatting is more complex than the simple word entry displayed above, we are also adding a `<link>` tag which imports CSS rules which are responsible for controlling how the generated content looks: transcriptions are bold and in a slightly larger font, forms and glosses render in the familiar “tiered” interlinear format, and so forth. With these three tags in place inside of an HTML page, and assuming that the `docling.js`, `education_in_jaro_sample-text.json`, and `docling.css` files are in the same directory as the HTML page itself, the rendered content on the right will appear. Note that once the web component has carried out

its rendering tasks, the single `<text-view>` tag is automatically populated with many additional “internal” tags to create structured, consistent markup. (In fact, the generated markup for this brief three-sentence example already contains 130 individual HTML tags.) Furthermore, the `<text-view>` web component enables time-aligned playback if the correct data is present in the associated JSON file.

Again, the component itself can be designed with particular uses and layouts in mind — the `<text-view>` shown here is merely a default possible rendering of a `Text` data as defined in the previous chapter. As we shall see in the next chapter, the same JSON data may be displayed with a variety of interfaces, appropriate to quite different documentary workflows.

3.3.1 A typology of web components for `docling.js`

Within the general notion of web components, `docling.js` components were designed in three “flavors”. This is merely a convention to try to keep the implementation of `docling.js` components as consistent as possible across the library. These three types of web components are *views*, *lists*, and *editors*. In the remaining sections of this chapter we will consider `View` components (which are used to display data objects) and the closely related `List` components, which are responsible for displaying arrays of objects. We will discuss the more complicated issue of designing and defining `Editor` components in the next chapter, but the following list will give an overview of some basic components for basic data documentary data structures.

Data	Component	Purpose
Word	<word-view>	Display a Word
Word	<word-list>	Display multiple Words
Lexicon	<lexicon-view>	Display a Lexicon
Sentence	<sentence-view>	Display an interlinear Sentence
Text	<text-view>	Display an interlinear Text
Sentence	<sentence-list>	Display multiple Sentences
Lexicon	<lexicon-editor>	Edit a list of Words
Sentence	<sentence-editor>	Edit a Sentence
Word	<word-editor>	Edit a Word
Text	<text-editor>	Edit an interlinear Text

Note that we adopt a simple, consistent naming convention for our three types of components, which are named first for the documentary data type that they address, and (after a hyphen) for which of our three types they implement.

Views

- <word-view>
- <sentence-view>
- <text-view>
- <lexicon-view>

As in the `<entry-view>` example above, View components generally take a single data object as input, and generate HTML markup which is appropriate for that data. In the current chapter, we will consider `<word-view>`s for rendering Word objects, `<sentence-view>`s for rendering Sentence objects, `<text-view>`s for rendering Text objects, and `<lexicon-view>`s for rendering Lexicon objects. Simple attribute values of objects are rendered as standard HTML elements — for example, a `<word-view>` will render its associated Word's form value, a string, inside of an HTML `` tag, and a `<sentence-view>` will render its associated transcription value into a paragraph (`<p>`) tag, and so forth.

Lists

- `<word-list>`
- `<sentence-list>`

Note however that as we saw in the previous chapter, some JSON objects have attributes where the *value* is in fact not simply textual content, but rather *another* JSON object. We can distinguish these values as being *simple* or *complex* — a Word object may have an attribute with the property `form` and a value which is simply a string (such as *bentana*). However, the attributes of some data objects may have values that are themselves not mere strings, but objects or arrays. Namely: a Sentence has an array of objects of the Word data type representing its glossed words; a Text contains an array of objects of the Sentence data type; and a Lexicon (like a Sentence) contains an array of objects of the Word data type. For cases where the value of a data object is complex, views will delegate their rendering

to a *list* component. Unlike view components, list components accept an *array* of objects as their data, and then automatically “stamp out” a view for each object in that array. Thus, a `<word-list>` component will automatically generate as many “child” `<word-view>`s as the array contains `Words`. List components are also useful as a target for defining search and sorting functionality.

Editors

- `<word-editor>`
- `<sentence-editor>`
- `<text-editor>`
- `<lexicon-editor>`

The most complex type of component in `docling.js` are `Editors`. Editor components provide an interface for editing data — either creating entirely new objects, or else by modifying existing objects: in terms of JSON data, we can say that editor components not only take JSON data as *input*, they also allow users to modify (or create from whole cloth) data which is ultimately *output* from the component. We shall see that editors are a prime target for customization, as many kinds of fieldwork workflows can be characterized as a sequence of editing steps. Note that the design and implementation of `Editor` components is very closely linked to *how* linguists go about carrying out the steps of their documentary work — because linguists use many different methodologies and techniques while working, it is important to bear in mind that the very same change from input to output data — that is to say, the very same dataflow — may be accomplished via entirely distinct

Editor components, each with their own user interface design. We will discuss this notion in Chapter 4. For more on the distinction between View (and List) components versus Editor components, see [§4.1](#).

3.4 Displaying data with view and list components

We will begin our discussion of the design of components in `docling.js` with the task of displaying data in a legible form for users. In Chapter 2, we used the JSON syntax as a convenient means of discussing abstract definition of data types. JSON is convenient not only as a way of conceptualizing data — `docling.js` components are designed to “accept” documentary data in the JSON format directly. To put it in general terms, when JSON data of the correct data type is “inserted into” to a `docling.js` component in a page, the component “knows” how to automatically render that data in a useful format.

Let us consider this process using our basic definition of a `Word` object, which is defined as a bundle of a `form` and a `gloss`, both of which are stored as string values in an object like the following sample Hiligaynon word. As we shall see, many other kinds of useful information *could* be recorded about this word beyond a simple `form` and `gloss`, but for the sake of simplicity we will consider only these two key fields in our exposition of `<word-view>`s.

First, we may render the word using the “abstract” tabular representation from Chapter 2:

form	eskwelahan
gloss	school

```
{ "form": "eskwelahan", "gloss": "school" }
```

And here is the equivalent data recorded in the machine-readable JSON data syntax:

```
{
  "form": "eskwelahan",
  "gloss": "school"
}
```

We see that this Word is defined as an *object* (it is contained in curly brackets { }) which has two comma-delimited properties, *form* and *gloss*, which are respectively associated with the values *eskwelahan* and *school*. (Recall that both properties and values are double-quoted, and are delimited by colons.)

But neither the abstract, but legible, tabulation above, nor the explicit, but poorly legible raw JSON syntax is familiar to linguists. We may compare a typical linguistic notation of the sort one might find in a glossary as something like the following:

<word-view>

eskwelahan ‘school’

This is of course just one possible typographic presentation of a word that might be found in a print dictionary — obviously dictionary entries in general may be far more complex, but simple “glossary” or “word list” formats of this kind are nonetheless quite common in published documentation. It is important to keep in mind that for a linguist reader, such a conventional typographic presentation is a data structure, in the sense that the reader knows how to map the typographical details onto categories of data. That this is the case can be shown by the fact that the conventions may be changed, without changing the core roles of each piece of data — choices about which typographic features should be used to indicate which data categories is largely a matter of aesthetics and convention. The table includes four examples which recreate the formatting of glossary-style “entries” in published word lists. (Entries have been simplified in some cases). Note that the permutations of bold or italic text differ between the examples. (In the case of the last example, no typographical variation is used at all.^{[14](#)})

Excerpted entry	Form	Gloss	Source
hraiwadubo turtledove	bold	(no styling)	Gothic (Lambdin 2006, 336)
āditta ‘burning, blazing’	(no styling)	‘quoted’	Pali (Gair and Karunatilake 1998, 182)
<i>sekqwet</i> ‘chipmunk’	<i>italic</i>	‘quoted’	Cupeño (Hill 2005, 136:470)
school hóhé	(no styling)	(no styling)	Eastern Kayah Li (Solnit 1997, 372)

Table 3.1 - Varied styling of forms and glosses drawn from published grammars

Again, for the purposes of digital language documentation, what matters is not so much that data be formatted in a particular way. What matters is that the categorization of data categories be captured in the content of the HTML document — and thus, within the user interface — in *some* systematic, unambiguous way. The `docling.js` components are designed to use only the minimum amount of default styling rules necessary to differentiate common classes of documentary data, while at the same time allowing for emulation of standard documentary notations. These defaults will be described below as we progress through the description of each component.

3.5.1 Granularity in digital documentation

Before preceding to the description of `WordViews`, let us set out a principle that has been applied in the implementations of all the components in the `docling.js`

library: the *granularity* of HTML markup. We will refer to an HTML document as “granular” to the degree to which every “piece” of the data it represents is “marked up” within the HTML managed by that component. For example, a component for displaying a `Word` object (per our working definition) will be “granular” insofar as it unambiguously encodes the “pieces” of an instance of the `Word` data type. Although our working definition of the `Word` data type is defined as having just two attributes, a `form` and a `gloss`, we will only say that a `Word` is “granularly marked up” if the HTML used to represent unambiguously encodes all *three* of the following “pieces” of data:

1. The `form` value
2. The `gloss` value
3. The `Word` as a whole

The last piece is crucial. A `Word`, after all, is a complex object, in the sense that it is a bundle of attributes which each have a name (`form`, `gloss`) and a value (“eskwelahan” and “school” in our example), but the bundling itself establishes a new entity which must also be marked up. We shall see examples of what consequences fall out from granular markup in discussions of specific components below. We may summarize the notion of *granular markup*, then, as follows:

GRANULAR MARKUP: HTML markup is *granular* insofar as (1) every object and (2) every value of every object in the data associated with the markup is displayed with a dedicated HTML element.

This principle has been adhered to throughout the implementation of

`docling.js` library of components. In the next section we will take a look at our first such `docling.js` component: the `<word-view>`. We will inspect the granular markup that the component manages directly, and discuss how that markup can serve as a flexible basis for designing interfaces for recording, modifying, and re-using words.

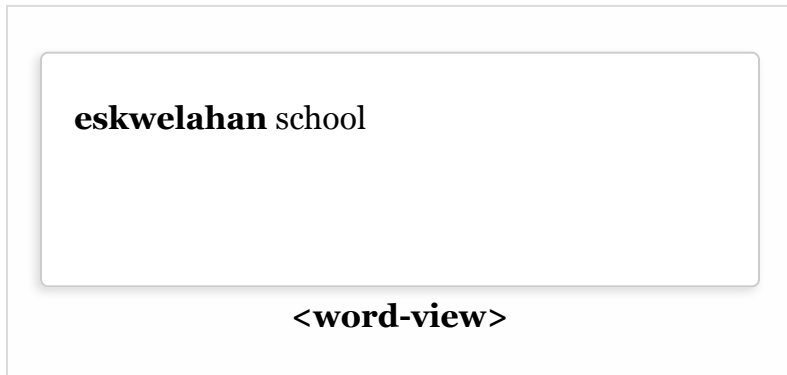
It should be borne in mind that the granular markup which is used inside `docling.js` components is not to be written “by hand” — the purpose of custom HTML elements is to generate appropriate markup for data without onerous hand-coding of HTML.

3.6.1 `<word-view>`

The first requirement set forth for a `<word-view>` above is that it unambiguously indicate what markup corresponds to a word object and its properties and values. There are many ways that this might be done in HTML, but in `docling.js` a custom element is defined to represent every data type in the documentary data model from Chapter 2. Note that custom HTML elements are identifiable as such by the hyphen in their tag-name. A `<word-view>`’s internal, generated markup looks something like the following:

```
<word-view>
  <span class="form">eswelahan</span>
  <span class="gloss">school</span>
</word-view>
```

Default styling rules for a `<word-view>` stipulate that the `` tag with a class of `form` is rendered with its font weight set to “bold”, resulting in the following presentation:



But there is more going on here than simple formatting of two strings of text. If you are reading this document on a device with a “pointing device” such as a trackpad or mouse, try moving your cursor over (“hovering over”) the words *eskwelahan* and *school* individually (on some mobile devices, clicking the words may work as well). The display of a status message *You are hovering over the form data* or *You are hovering over the gloss data* has been programmed to appear as a function of your cursor’s location over those rendered `` tags. Furthermore, each “hovered” stretch of text is highlighted with a distinct background color, as a function of the type of data which that text conveys.

This simple example displays one of the most crucial advantages of the web platform as an environment for digital language documentation: Granular markup is

the basis for both modifications in visual presentation, and for customized, programmatic responses to user behaviors. Those stylistic and behavioral modifications can be applied selectively to individual elements or to sets of elements. There is an entire “language” (in the sense of “syntax for formatting text in a machine-readable way”) called CSS (*Cascading Style Sheets*) for handling the selection of elements in documents and modifying the way they are presented. We will introduce some of the basic concepts of CSS as we progress through components in this chapter.

Note that the “hover” behavior in the previous example was a customization of that particular `<word-view>`. The particular hover behavior (namely, displaying a message and changing background colors) is not part of the generic, “base” functionality of a `<word-view>`, and consequently other instances of the `<word-view>` component in the document do not exhibit those behaviors. This customized interaction is not particularly useful or reusable, but it demonstrates how components may be customized to change how data, markup, and user interactions interact. In practical terms, the possibilities for how such interactions might be defined are practically limitless. We shall see how such components may be combined into full applications for carrying out key documentary tasks.

It is also worth noting the way in which these two `` elements are displayed: they “run together” on a single line. As we shall see below, this style of laying out the content of the two spans of text can be overridden such that they are arranged vertically in a column, as would be appropriate in the context of an interlinear gloss.

3.7.1 <sentence-view>

Moving on now to a more complex `docling.js` component, we may consider the `<sentence-view>`. The data representing a `Sentence` is more complex than that for a word, in that it contains (at least) `transcription` and `translation` attributes, as well as a compound value for its `words` attribute: an array of `Word` objects. In Chapter 2 ([2.3.1](#)), we saw how the JSON data syntax represents objects via “curly brackets” (`{ }`), and arrays are delimited by square brackets (`[]`). The value of a `Sentence`’s `words` attribute is an array of `Word` objects, and thus contains a sequence of comma-delimited objects inside an array. Here, for example, is JSON representation of the first sentence object from the sample Hiligaynon text:

```
{
  "transcription": "Hello, ako si
Juan Lee.",
  "translation": "Hello, I'm Juan
Lee.",
  "words": [
    {
      "form": "hello",
      "gloss": "hello"
    },
    {
      "form": "ako",
      "gloss": "1S.ABS"
    },
    {
      "form": "si",
      "gloss": "PERS"
    },
    {
      "form": "Juan",
      "gloss": "Juan"
    },
    {
```

```

    "form": "Lee",
    "gloss": "Lee"
  }
]
}

```

JSON data representing a sample sentence which will be used in the discussion below.

The tabulation format from Chapter 2 may also be used for a simpler visualization:

transcription	Hello, ako si Juan Lee.	
translation	Hello, I'm Juan Lee.	
words	form	gloss
	hello	hello
	ako	1S.ABS
	si	PERS
	Juan	Juan
	Lee	Lee

An example of the Sentence data type, as JSON and tabulation
 Extracted from [education_in_jaro-text.json](#)

As both the “raw” JSON syntax and the tabulation visualization demonstrate, this simple Sentence object contains two string attributes — a transcription and a translation, and an array of Word objects, each with its constituent form and

gloss. This data structure has the benefit of being machine-readable: a computer can “understand” the structure of the JSON file.¹⁵ But while JSON and abstract table representations like those above have been helpful in our own conceptualization of the data, at this point we will turn away from thinking strictly in terms of data formats, and toward the notion of how to form *user interfaces* for interacting with data in a more usable way. We wish to present data of this (and other) documentary data types using notations which are already familiar to documentary linguists. It is the responsibility of the `docling.js` components to do just that: a `docling.js` component defines a “piece of user interface” that maintains an association between the data and the display (and which can be combined with other such components). In the remainder of this section we will investigate how granular markup generated by the `<sentence-view>` component leads to good usability characteristics for as many users as possible.

Granularity and responsiveness in the `<sentence-view>` component

We saw an example of this in the discussion of `<word-view>`s above, which manage the relationship between `Word` objects and granular HTML markup. But when we turn to the `Sentence` data type, it becomes clear just how important it is that the relationships between data and user be maintained correctly. It is not *just* a matter of making the presentation look “right” or “familiar”, it is also a matter of ensuring that the markup generated by a component is granular.

The consequences of granular markup (and the lack thereof) can be seen in the demonstration below. Consider the two following, seemingly near-identical

renderings of the example Sentence data:

<word-view>s Small Large Enumerate

Display One

Hello, ako si Juan Lee.

hello	akó	si	Juan	Lee
hello	1S.ABS	PERS	Juan	Lee

Hello, I'm Juan Lee.

Display Two

Hello, akó si Juan Lee.

hello	akó	si	Juan	Lee
hello	1S.ABS	PERS	Juan	Lee

Hello, I'm Juan Lee.

Two displays of the data above, with differing underlying markup.

These two presentations are nearly identical visually, and adhere to the familiar “four-tier” interlinear text format. As described in style guides for major linguistics journals (such as LSA ([2021](#)) and IJAL ([2021](#))), this format consists of four “tiers”: (1) a transcription line (or “baseline”); a list of morphologically glossed words, each represented by (2) a form set in bold text together with a (3) corresponding gloss; and finally (4) a free translation (in this case, into English). In the “three-tier” format, the initial transcription line is dispensed with.

But the underlying markup of these two displays is distinct in a crucial way —

only Display Two is granular in the sense defined above. We can get a feel for the importance of granularity here by interacting with the display, and modifying its visual characteristics.

Resizing the <sentence-view>'s container

One such interaction is the way that the content reacts to the size of its container. To see this, the “resize handle” in the lower right corner of the demo above may be used to adjust the width of the demo. Each display responds distinctly when the containing box is made smaller than the default width of the content: in that context, the content of the first interlinear is obscured — it is said to “overflow” in CSS jargon — and it becomes partially obscured from view, and thus unusable. However, the second interlinear “responds” to the resizing, with each <word-view> “wrapping” onto a new line as necessary, without disturbing the internal vertical arrangement of its form and gloss.

Increasing font size

A similar change can be effected by adjusting the font size of the interface via the *range* input (or “slider”) to the largest size. As the font size increases, the user interface also adapts in the second display, but not the first. Note that changing font sizes may be more than a convenience for users with limited vision: it may be the only way that they can interact with visual documentation on a screen.

It may help to visualize this behavior by toggling the borders of each <word-view> via the button labeled *Toggle <word-view> borders*. A red border <word-view> will appear around each word, with its two internal tiers. That it is possible in Display Two to “target” the individual <word-view>s in this way is a simple consequence of the fact that there *is* an element which corresponds to the word level of analysis in that display: it is granular markup. In the first display, there are no elements corresponding to each word as a whole.

The third button is perhaps more informative, in that it enumerates the HTML elements corresponding to the `form` and `gloss` values in the order in which they appear in the HTML source. Note that in the first display, the numbers proceed *across* each row, whereas in the second row, they proceed first *down* the tiers, and *then* across.

The example Sentence object’s three properties are rendered in accordance with the type of data their values contain. The `transcription` and `translation` properties are rendered in a manner which is analogous to the way the `form` and `gloss` properties of the Word object was rendered in the <word-view> component: each is displayed within a standard <p> (paragraph) tag. The second element is a <word-list>, which is responsible for rendering the Sentence’s individual

<word-view>s. which we will explain in the next section. Here is part of the resulting markup:

```
<sentence-view>
  <p class="transcription">Hello,
akó si Juan Lee.</p>
  <word-list></word-list>
  <p class="translation">Hello, I'm
Juan Lee.</p>
</sentence-view>
```

Generated HTML markup in a <sentence-view>
(Contents of the nested <word-list> are
explained below)

Notice that unlike the and in the <word-view> above, when the <p> tags here are displayed, they each are assigned a new line, whereas the s marking up *eskwelahan* and *school* ran together onto a single line. This distinction is referred to as the *display* value of the HTML elements: <p> tags are said to be *block-level elements* and tags are said to be *inline elements*. That the display of individual elements can be addressed and modified in this way is one reason we can say that the web platform separates

logical structure from presentation.

Accessibility is a universal concern

So far, we have discussed these variations in markup as though the visual modality were the only means by which users would want to access this content. And indeed, the fact that we can define markup which adapts to available space means that we can create web content which can adapt to physical screens of differing sizes. In web design circles, such markup is referred to as *responsive* (Allsopp ([2000](#)), Marcotte ([2010](#))). Responsive design is a crucial consideration for web-based documentation, since we must bear in mind that the use of mobile devices continues to increase rapidly. We should expect that the proportion of linguists who access language documentation via the web (especially in communities with limited bandwidth availability) will continue to increase with time. Non-responsive formats such as PDFs are of limited utility on mobile devices.

But responsive design is just one aspect of a much more consequential, and critical, concern: *accessibility*. Visual accessibility (in the contexts of varying device size and the limitations of vision for some users) probably affects most users of the web. But we should not limit our concern to those users. Indeed, the notion that the web should be a universal means of accessing information is one of its original design principles:

The Web is fundamentally designed to work for all people, whatever their hardware, software, language, location, or ability. When the Web meets this goal, it is accessible to people with a diverse range of hearing, movement,

sight, and cognitive ability.

Thus the impact of disability is radically changed on the Web because the Web removes barriers to communication and interaction that many people face in the physical world. However, when websites, applications, technologies, or tools are badly designed, they can create barriers that exclude people from using the Web.

Initiative (WAI) ([n.d.](#))

Building documentation with the web platform enables many avenues for the development of accessible content. Responsiveness to a variety of device types is just one of these capabilities. The basic principle of granularity in markup advances this goal. We shall see more examples of accessibility in subsequent sections.

3.8.1 <word-list>

We saw some examples of rendered <sentence-view>s in the previous section. Between the <sentence-view> and its contained <word-view>s, there is an intermediary element called a <word-list>. In the display below, identical <word-list> components are shown in two distinct contexts: first (left), within a <sentence-view>; and second (right), within a <lexicon-view>.

```

<sentence-view>
  <word-list>
    <word-view>
      <span class="form">hello</span>
      <span class="gloss">hello</span>
    </word-view>
    ...additional word-views...
  </word-list>
</sentence-view>

```

```

<lexicon-view>
  <word-list>
    <word-view>
      <span class="form">hello</span>
      <span class="gloss">hello</span>
    </word-view>
    ...additional word-views...
  </word-list>
</lexicon-view>

```

hello	akó	si	Juan
hello	1S.ABS	PERS	Juan
Lee			
Lee			

hello hello
akó 1S.ABS
si PERS
Juan Juan
Lee Lee

The same <word-list> within a <sentence-view> and a <lexicon-view>

We will look at the <lexicon-view> in more detail in the next section, but note the difference in display characteristics within the two contexts: in the <sentence-view> context, the form and gloss spans display as described in the previous section, stacked vertically and wrapping across lines, whereas in the <lexicon-view> context, the layout corresponds to what we saw in the description of <word-view>s: a “one-word-per-line” layout of the kind found in simple dictionaries. The actual content of this “lexicon” looks rather peculiar here, in fact, since in most cases we would expect to see such a display with sorted “entries”.

Roughly speaking, these differences in display are controlled by *selecting* the nested <word-list> components and applying distinct style rules which toggle the display property of CSS. The display property is quite powerful, and is used to enable the distinct renderings of words in these two contexts. (For more details, see

“Display - CSS: Cascading Style Sheets” ([2021](#).) Without delving too deeply into the mechanics of CSS, we may note in passing how the two relevant CSS rules are defined:

```
sentence-view word-list word-view {  
  display: inline-grid;  
}
```

```
lexicon-view word-list word-view {  
  display: block;  
}
```

The sequence `sentence-view word-list word-view` means “select all `<word-view>` elements which are nested with `<word-list>`s and then `<sentence-view>`s.” The specific property of `display` is then set within the curly brackets. Briefly, the `display` property here is set to the value `inline-grid`, which instructs the element to behave like an inline element — that is, to sequence themselves like words in a line of text — but for the *content* of the element (that is, the paragraphs containing the `form` and `gloss` values) to “stack” vertically. The second rule works similarly, selecting those `<word-view>`s which are (grand-)children of a `<lexicon-view>`, and then carries out essentially the opposite configuration: each `<word-view>` gets its own line, but the `form` and `gloss` values run together on the same line. The `docling.js` has been designed with default presentations of familiar documentary content in mind. All such defaults may be overridden by modifying the defaults, or adding completely new CSS rules.

Despite the quite varied appearance of these elements, the feature of making the same “logical” structure (in this case, a `<word-list>` which contains constituent `<word-view>`s) render differently in different contexts actually helps to keep the system simple. A `<word-list>` component isn’t defined to “check” which context it is appearing in (a `<lexicon-view>` or a `<sentence-view>`) — it is implemented with the single, simple responsibility of generating `<word-view>`s for an array of `Word` objects.

3.9.1 `<metadata-view>`

Thus far we have been considering fairly small, isolated quantities of data — `Words` and `Sentences`. But as we begin to delve into displaying the instances of the `Lexicon` data type, we encounter a common problem in documentation: how to keep track of data *about* data, that is to say, metadata. Before proceeding to the discussion of the `<lexicon-view>` itself, we will briefly survey the `<metadata-view>`, a simple presentation of arbitrary inline metadata that may be included by default in `<lexicon-view>`s, `<grammar-view>`s and `<text-view>`s.

While metadata may be added to any of the documentary data types in the `docling.js` library, by default the `Lexicon`, `Text`, and `Grammar` require at least minimal metadata be included alongside the data itself. Note that there it is only one required field: `title` — otherwise, there are no specific requirements as what metadata fields should be present, nor how those fields should be arranged, beyond the fact that it should be possible to represent that data as a JSON object. We briefly introduced this notion of metadata in our discussion of various data types in the data

type catalog [2.3.1](#) in Chapter 2. This design is based on the premise that differing documentation projects (and archives) require different kinds of metadata, and that `docling.js` is not intended as a metadata standardization mechanism.

To render these metadata objects, we define a `<metadata-view>` which can render a metadata object regardless of its structure. In the case of lexicons, such metadata may contain essentially any kind of annotation, but some simple defaults might include a name for the lexicon, details about what language or languages it contains, information about the contexts in which it was recorded, and so forth. Note that the `<metadata-view>`s are responsible only for *rendering* such metadata, not for *editing* it.

Below are three example `<metadata-view>`s populated with metadata drawn from various kinds of data in the `docling.js` JSON format.

▼ Metadata...

name	Rapa Nui
family	Austronesian
source	Du Feu (2012)

Sometimes even the simplest metadata is useful for keeping track of the source of an object. Here is metadata associated with Language object representing Rapa Nui. This metadata is so minimal as to provide little more than a reminder of where the information was drawn from.

▼ Some Esperanto Proverbs

title	Some Esperanto Proverbs
source	Wikiquote.org
id	esperanto_proverbs
url	https://en.wikiquote.org/wiki/Esperanto_proverbs
license	CC BY-SA 3.0
language	Esperanto
notes	Morphological analysis by Patrick Hall

Sources may provide particular kinds of information that is worth recording in a structured way. Here, a text collecting Esperanto proverbs derived from wikiquote.org has been annotated with licensing information and brief note on a particular kind of annotation added.

▼ 5. Marriage and Divorce

title	5. Marriage and Divorce		
citation	Eatough, A. 1999. Central Hill Nisenan texts with grammatical sketch. Univ of California Pres. p.84		
	type	url	citation
	audio	marriage_and_divorce.mp3	A monologue about betrothals, in-laws, and divorce, LA 40.008, in "The Richard Smith collection of Nisenan sound recordings", Survey of California and Other Indian Languages, University of California, Berkeley, http://cla.berkeley.edu/item/14969 .

This metadata from a Central Hill Nisenan text includes metadata drawn in turn from the California Language Survey, including a citation as per that archive's standards and Links array containing the file name of a local copy of associated media files.

Sample metadata structures with corresponding JSON data

The question of standardizing or normalizing metadata structure is a complicated one: often archives specify particular metadata schema to be used in deposits. Here, we are taking a pragmatic approach, specifying only that certain data types (at least Text objects, Lexicon objects, and Grammar objects) should have an “inline” metadata structure — that is to say, basic identifying metadata of some kind should *always* be included in the same file as the data itself. This approach avoids the common frustration of a discovering a “metadata-less” data file whose origins are unknown. In many cases such data may become unusable, if, for instance, it is separated from a metadata spreadsheet.

3.10.1 <lexicon-view>

In [§2.3.1.6](#) we described the Lexicon documentary data type, which is used to store a list of unique words. In the current section we see an implementation of a user interface for displaying Lexicons, the <lexicon-view> component, whose design we will now consider. Both <lexicon-view>s and <word-list>s manage a list of Word objects. However, the <lexicon-view> has a more complicated list of responsibilities than simply iteratively rendering its list of words. Every <lexicon-view> is responsible for providing a user interface which allows the user to carry out three key tasks in interacting with lexical data: 1) view associated metadata, 2) sort the words in the lexicon according to various criteria, and 3) search the lexicon according to various criteria.

Sorting

The order in which words are displayed can vary depending on the workflow in which the <lexicon-view> is being used. As a simple example, a <lexicon-view> might sort its words by form (the typical view), or by gloss (a “reverse” lexicon). We shall see this control in context in the <lexicon-view> example below.

Search

Search, or filtering of the words in a lexicon, is key functionality of <lexicon-view>s. (Note that explicitly carrying out a search through a query interface (of the sort found in an internet search engine, for instance) is only one means of searching

a <lexicon-view>. As we shall see in the section on <text-editor>s, below, the search functionality in a <lexicon-view> may also be programmatically used in semi-automated glossing.)

User interaction with these responsibilities is carried out through a set of controls in the “header” of the view. Typically, the header is rendered above the <word-list>. Here, then, is a basic rendering of a very short lexicon:

▶ A few Hiligaynon words
▶ Search
Sort on:

akon 1S.ERG2
akón 1S.ERG1
amon 1PL.ERG1
CPU CPU
dirá there
inyo 2P.ERG1
knowlédge knowledge
ko 1S.ERG1
kwa’a get
maéstra teacher(F)
mga PL
mo 2S.ERG2
na link
namón 1PL.ERG2
nila 3PL.ERG2
nu know
nya 3S.ERG2
nyo 2PL.ERG2
sa to
si PERS
sila 3P.ERG1

<lexicon-view>

The sample data here were chosen purposefully: many of the glossed form in this small lexicon contain the label ERG1 or ERG2. I frankly do not recall why I used these

labels at the time. This is down to my own poor practice in annotating my documentation at the time I created it, but a similar situation can arise when a linguist is trying to gain understanding of earlier work, which is sometimes unpublished or even in manuscript form.

The very basic default implementation of search in this interface does permit wildcard searches, which would be relevant in trying to recover an understanding of what these terms mean. For instance, searching for the query `*ERG1` in the `gloss` input will turn up forms that end with the grammatical category label `ERG1`. More likely, we would want to try to begin comparing `ERG1` and `ERG2`, so in that case one approach would be to use a string search like `*ERG*` over the `gloss` field. This at least serves to begin filtering the data.

String searches as these, however, can only take us so far. In the next chapter we will see more nuanced search interfaces which include more granular conceptions of grammatical categories and linguistic units like morphemes. For now however let us turn to the components necessary for rendering texts.

3.11.1 `<sentence-list>`

The `<sentence-list>` component is analogous to the `<word-list>`, but rather than rendering an array of `Words` via a sequence of `<word-view>`s, it renders an array of `Sentences` with a sequence of `<sentence-view>`s. Like the `<word-list>`, `<sentence-list>`s typically only occur within a parent view, usually a `<text-list>`, as we shall see in the next section.

3.12.1 <text-view>

At this point we have seen the necessary components to build up a view of a `Text` object. In the sample extract below, we see a basic demonstration of the component. The default view contains a **metadata** section, a **search** section, and then the main **content**, consisting of a <sentence-list> which in turn renders <sentence-view>s.

Sample of “Education in Jaro” text (sentences 1-5)

► Education in Jaro
► Search

▶ 0:00 / 4:35 🔊

▶ 8.580-9.650 **Hello, akó si Juan Lee.**

hello	akó	si	Juan	Lee
hello	1S.ABS	PERS	Juan	Lee

Hello, I’m Juan Lee.

▶ 9.704-10.714 **matopic na akó subóng**

ma-topic	na	akó	subóng
IRR-topic	already	1S.ABS	now

I’ll start the topic now

▶ 10.753-11.818 **parte sa mga eskwélahan.**

parte	sa	mga	eskwélahan
part	to	PL	school

The structure of the `Text` data type is more complicated than that of a `Lexicon`, given that `Texts` contain an array of `Sentence` objects, which themselves contain

an array of `Word` objects. In practice we want to take advantage of this granular representation, since we might want to carry out various kinds of search across the `Text` object.

At this point we have surveyed the most important data types likely to be necessary for recording documentary data in a fairly transparent and maintainable way. Thus far, however, the discussion has been about data in abstract terms — comparatively little has been said about the way in which such data should be *produced* and *used*. Of course, directly editing large JSON files is certainly not a feasible approach to digitizing documentation. Instead, in the next two chapters we turn to the question of user interfaces and how they relate to linguists' workflows. In Chapter 4 we consider how to *display* (or “view”) such data, and in Chapter 5 we broach the more complex topic of how to *edit* it — a task which is deeply intertwined with the varied nature of our documentary practices and workflows.

4. Components for documentation workflows

4.1 Designing user interfaces for documentation workflows

In Chapter 1 we discussed a generic approach to specifying a *dataflow*, a *workflow*, and a *user interface*. We saw how a particular dataflow (transformation of input data into output data) could be carried out via various kinds of fieldwork “workflows”. Chapter 2 described how data types (such as Words, Sentences, Texts, and Lexicons) could be designed as an abstract data structure, which in turn could be encoded with the JSON notation. Chapter 3 addressed the presentation of such data in the form of “web components” — custom HTML elements designed specifically to display documentary data types.

But displaying data is only a small part of the much larger problem of creating user interfaces for managing documentary data. Beyond simply *displaying* data, we also need to address the tasks of *creating* and *editing* data. The diagram below draws this distinction with two flowcharts. The flowcharts depict how data flows through a component. In the upper chart, data is depicted as “flowing into” (or being loaded into) a *View* component. The *View* then generates an appropriate presentation of that data (typically as HTML markup) to the user. As far as data is concerned, the *View* component’s responsibilities are simple: it serves only to display data.

In the lower flowchart, representing *Editor* components, data flows not only *in*, but also *out*. Like *View* components, *Editor* components must generate a legible

user interface in the form of HTML markup, but that generated markup will include user interface elements which the user can control. The user (the linguist) can use those controls to insert data into or edit data in an `Editor`, as opposed to simply viewing pre-existing data in the manner of a `View` component.

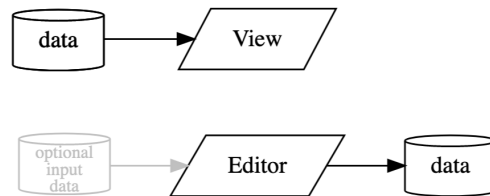


Figure 4.1 - Basic Docling application patterns

An `Editor` component is designed to enable editing of a particular data type. Returning to our now-familiar `Word` data type, a `<word-editor>` component could provide a means of editing a `form` and a `gloss`. As shown in the gray input element in the flowchart, Editors may optionally accept existing input data — typically data which is only partially complete, such as an array of `Word` objects with `gloss` values but with empty `form` values. (A `Swadesh` list is data of this kind.) As we shall see below, an `Editor` may also be designed to create data “from scratch,” without any pre-specified values.

From one viewpoint, “digital language documentation” can be conceptualized as a series of steps which progressively carry the documentation closer toward an ever-more developed `Boasian Database`: as collection and recording instances of many different data types progresses. But to understand fieldwork merely as the collection of structured data is far too simplistic. All working linguists know that fieldwork is not a matter of marching up a hierarchy of data types, one after the other, from

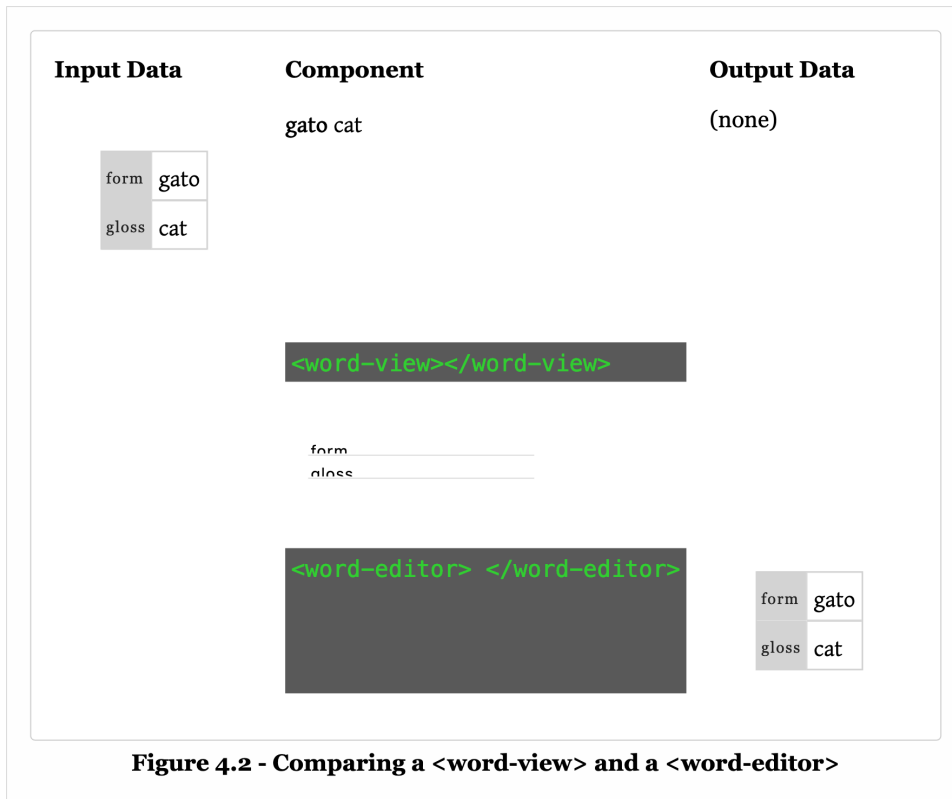
phonemes to discourse, building up tidy, complete data structures as we go. It is true that the data model of the “Boasian trilogy” as introduced in Chapter 2 is reasonably simple and fairly complete, at least insofar as “raw” documentary data is concerned.

But in practice, fieldwork is a messy affair. Fieldwork requires jumping back and forth between the whole set of data types: progress is often shaped in a conversational and sometimes even happenstance manner (a sought-after minimal pair turns up during the elicitation of a narrative; an unusual syntactic construction appears in an offhand comment while investigating phonology, etc.). Our goal here is not to attempt to somehow standardize fieldwork workflows into some hypothetical “logical” recipe for building up documentation. To the contrary, we wish to embrace the way that documentary linguists adapt to the changing interests and concerns of both the research itself and the people who are participating in it: our goal is to develop interfaces that support the varied nature of documentation processes.

As we saw in Chapter 1, where we considered the implications of thinking of including “record” buttons within a user interface for a word-collection workflow, there are many ways that linguists might go about carrying out particular documentation tasks.

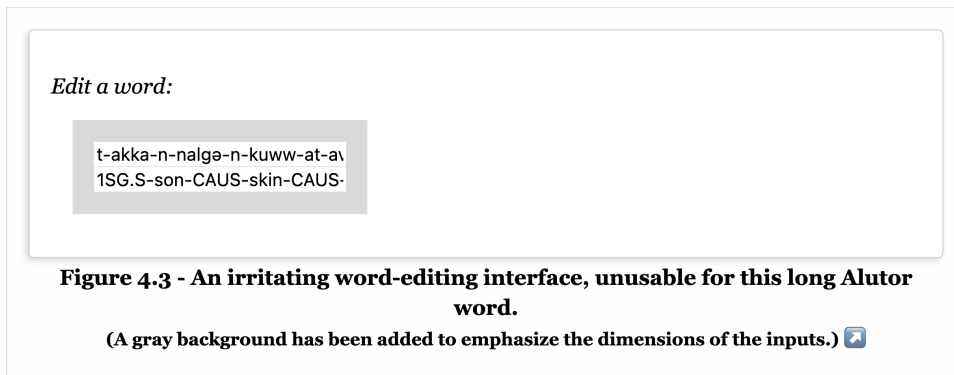
It is tempting to imagine that we could define a tidy set of Editor components that paralleled the basic View components we discussed in the previous chapter exactly: in such a scenario, a `<word-editor>` would be written for editing Words, a `<text-editor>` for editing Texts, and so forth. There is in fact some utility to creating such “default” editors for particular data types. In the display below, a `<word-view>` component appears next to a very simple `<word-editor>`

component. The instance of a Word data type displayed by the `<word-view>` has two properties, a form and a gloss, and just so, the `<word-editor>` provides *inputs* for a form and a gloss.

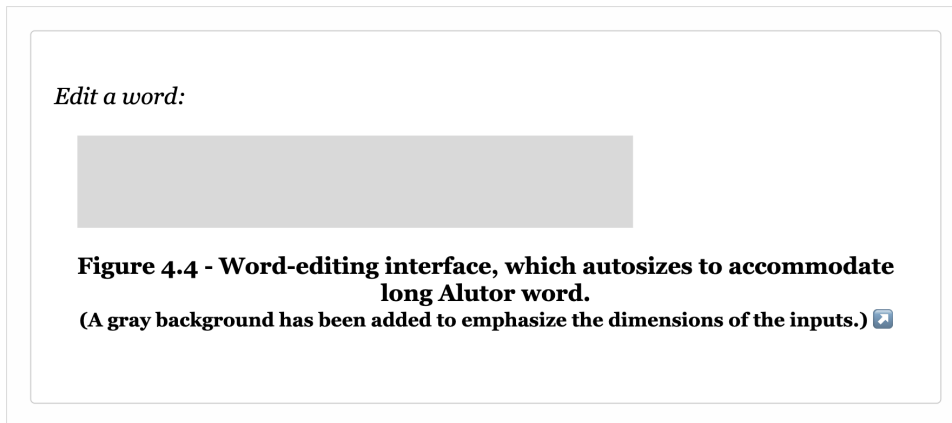


This approach is fairly easy to understand in simple cases, but it glosses over the complexities of real documentary workflows. It is difficult to define what we mean by “editing” in a general way, even where the data being edited has a straightforward structure, as it does here. Consider, for example, how language typology might play in to the utility of a simple word-editing interface. The simple `<word-editor>` component displayed above is a custom component, but it is itself composed of “default” HTML elements, specifically, the standard `<input>` element, which was introduced in [§1.3.4](#).

Standard HTML elements have presentational defaults, such a default width. But many languages (particularly agglutinative and polysynthetic languages) have words that are very long indeed, far too long to fit legibly within the default width of a standard HTML `<input>`. For example, it would be quite frustrating to try to enter a complex form from a polysynthetic language — exemplified here by the Alutor (Chukotko-Kamchatkan) word **t-akka-n-nalgə-n-kuww-at-avə-tk-ən** ‘1SG.S-son-CAUS-skin-CAUS-dry-SUFF-SUFF-PRES-1SG.S’ ([Gerdtz 1998, 92](#)). As shown in the pre-filled form below, these inputs are by no means sufficient to input such a word:



Editing such a word results in a horrible user experience: it is impossible to see the entire word at once! ¹⁶



Workflows to transcribe individual words in other languages might encounter no such difficulties — “isolating” languages which tend to have monomorphemic words tend to have short words, for which an unmodified HTML `<input>` tag with the default width might suffice. In this work we advocate the stance that we should not attempt to create “the” word-editing interface. Instead, we should think about how a user interface *could* assist in the process of carrying out a workflow — we should seek to develop a *design process* for creating editable user interfaces which meet our working needs.

To that end, this chapter is not organized like the previous one, where we surveyed View components in a roughly hierarchical sequence of increasingly “larger” data structures, from Words, to Sentences, to Texts, from Words to Lexicons, etc). Rather, the sections of the current chapter are based on specific, actual workflows that linguists might use as they are carrying out fieldwork. As we shall see, the way in which the content of data structures is actually built up in fieldwork can vary drastically depending on the workflow a linguist is using. We seek to encourage support for a variety of workflows, not to constrain or standardize workflows merely for the sake of uniformity in practice.

We also describe the process of *composition*, whereby composite user interfaces for complex workflows, handling complex data structures, are composed from simpler components. Ultimately, it is not possible to enumerate all the ways that linguists go about collecting data. However, reusable patterns will emerge. Again, we emphasize the idea that designing interfaces for documentation is a design problem: we must equip ourselves with a vocabulary for discussing how documentary workflows and user interfaces should relate in *particular* contexts.

4.2.1 Static documents vs. interactive interfaces

The interpretive nature of transcription has long been a topic of discussion in language documentation as well as linguistics at large. Ochs ([1979](#)) is a seminal article on this topic, having been repeatedly cited in the literature on language documentation (see Seidel ([2016](#)), Himmelmann ([2012](#)), Bird ([2021](#)), and Meakins, Green, and Turpin ([2018](#)) *inter alia*). Ochs was concerned with the relationship between transcription — the written record of speech events — and information.

One of the important features of a transcript is that it should not have too much information. A transcript that is too detailed is difficult to follow and assess. A more useful transcript is a more selective one. Selectivity, then, is to be encouraged. But selectivity should not be random and implicit. Rather, the transcriber should be conscious of the filtering process. The basis for the selective transcription should be clear. Ochs ([1979](#)) p. 167

The use of digital representations of linguistic data was by no means novel in 1979 — even in 1967, Samarin could state that the “linguistic use of electronic

computers already has a relatively long history” ([Samarin 1967, 170](#)) — but Ochs was working in a world where creating and using digital corpora remained a time-consuming and expensive undertaking: there was therefore little room for “experimenting” with the presentation of data. This “formatting debt” enforced a kind of uniqueness on transcriptions: committing to one way of arranging the print transcription was not a trivial, transparent matter of “formatting”. It was a consequential decision about what the permanent, canonical representation of that speech should be. Deciding how to present data in print is a decision to be made with great attention, Ochs argues, because the presentation could influence or even constrain the reader’s interpretation.

Ochs’ example data consisted of two-column transcripts of interactions between adults and children — her domain was child language acquisition — each speaker was arranged in one of two columns. She suggested that decisions about the ordering of those columns, i.e., whether the child’s speech first or second — subtly modulated the way that a reader would interpret the data. If the adult was placed in the left column, she suggests, the child would be interpreted in a more subsidiary or reactive role. If the child were first, it was the adult who would be taken to be reactive. But such decisions are far less categorical and final in the digital domain. Digital representations of data are malleable, and formatting of data is actually *less* consequential, precisely because the display of data can be itself be selective and even interactive: the reader can be given the option to re-arrange the display in more than one way.

Let us consider an interactive display of data which is comparable to Ochs’, but

also more oriented toward language documentation than child language acquisition *per se*. We will make use of a brief excerpt from the Murrinhpatha Language Acquisition Project (LAMP) ([Forshaw 2016](#)).

Turning then to the example problem of the ordering of columns in a child-adult dialog transcript, below we present a digital interface in which the order of columns is modifiable. Choosing one of the three “radio button” options in the header of the excerpt below allows the user to toggle the display between child-first, adult-first, and left-aligned views.

Figure 4.5 - Interactive dialog text view
toggle radio buttons to modify layout

☒ adult-first ☐ child-first ☐ left-aligned

<small>CARLA ADULT</small> orange nanherdharra orange oranges you go get oranges for us	<small>ACACIA CHILD</small> wawit oranges
<small>CARLA ADULT</small> yu yes	<small>ACACIA CHILD</small> xxx xxx xxx xxx
<small>CARLA ADULT</small> orange nanherdharra you go get oranges for us	<small>ACACIA CHILD</small> wawit ngay-yu mama mum orange me
	<small>ACACIA CHILD</small> wayit ngawu two two oranges me

This small interface demonstrates a distinct advantage of digital representations of documentary data: displays may be designed to be not just selective — as Ochs recognized all (typeset) documents must be — but selectable *interactively*. In the previous demonstration, a single data structure (a text where each sentence is annotated with metadata indicating the age of the speaker) can be rendered and re-rendered. Interface controls (a simple radio-button interface here) allow the user to make decisions about how to arrange the contents of the interface.

We shall see below how flexible interaction with data is a key consideration when designing user interfaces for specific language documentation workflows. If we begin the task of designing a user interface by asking how we *want* to interact with documentary data itself, we are much better positioned to effectively support our working methods, as opposed to always adapting them to the range of software interfaces that happen to exist.

4.3 Transcription

A personal aside on digital transcription

Like many linguists, I tend to romanticize the raw materials of fieldwork: notebooks cast a particularly strong spell. Writing notes in notebooks is, quite frankly, fun.

However.

One day in my graduate fieldmethods course, in a “one-on-one” session with Joshua De Leon, our marvelous teacher of Hiligaynon, I was taking notes in one of my notebooks. A Moleskine with a plain brown cover and no lines and a spine that lays flat, thank you very much. And I had to use Pilot G2 black ink fine-tip pens. For later corrections, I would switch to a red G2, but it had to be extra fine...

I had asked Joshua a question, and as always he had responded with enthusiasm, humor, and high-speed Hiligaynon. I started scribbling an attempt at a transcription of his reply. I wanted to get it right, and it was taking some time.

Joshua leaned across the table. “Pat!” he said, finally. He was tired of waiting for me to write.

I looked up.

*“Why don’t you just **type** it?!”*

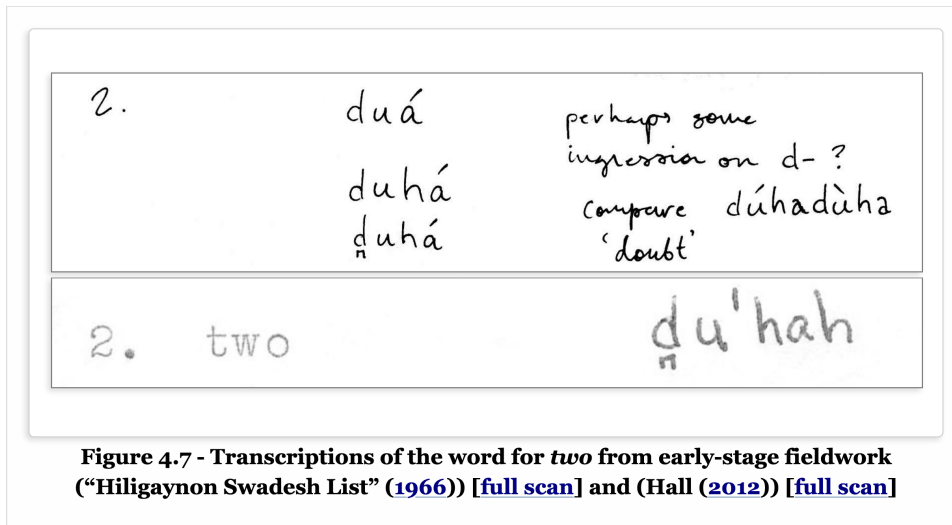
I will not forget the look of disbelief on his face. We were, after all, in an air conditioned university classroom, surrounded by technology. A notebook was the wrong tool for that job in that context. (It remains an appropriate tool in other contexts.)

I realize that many linguists have the opportunity to work in locations that are remote from conveniences such as electricity, without reliable access to computers, and they must continue to rely

on written notes. I still cherish my old notebooks, but if I could find the time I would digitize them all immediately. I no longer believe that manuscript transcriptions are inherently more flexible than “born-digital” transcriptions.

And at that moment, I opened up a text file, and typed his utterance. I closed my notebook for good. The new text file presented its own problems, but at least it was searchable. It was a start.

As a first design problem, we will consider challenges surrounding the fundamental task of transcribing utterances in a phonetic alphabet. Transcription is problematic in a digital context, and particularly so in the early stages of fieldwork when it is not clear whether a given perceived phonetic feature is significant (or even really present). Linguists err on the side of narrower transcription in early stages, where possible, but a narrower transcription style brings its own challenges. Most glaringly, it requires more symbols, and this in turn requires a means of inputting those symbols. Many approaches to handling phonetic input have been suggested. There are many precedents for web-based tools, mostly in the form of “clickable” IPA charts, including Szynalski (2021), Ishida (2021), and Ruter (2021), and the International Phonetic Association’s own “i-charts” “IPA i-Chart” (2021). Most of these tools take advantage of the familiarity of the International Phonetic Alphabet to linguists, by providing the same visual layout as the standard charts, but programming the interface such that clicking the character inserts it into a text field. Others are based on the graphical similarity of some characters (Szynalski (2021)), so that «β», «6», and «B» are grouped together.



These interfaces are quite useful for certain tasks, but from our perspective they also have limitations. One problem is that they are not modular: they are not easily integrated into other user interfaces, and as Bower (2015) points out (p. 40), “as soon as you are typing even small amounts of data extra keystrokes or mouse clicks slow down data entry considerably.” Imagine, for instance, trying to insert IPA characters via one of these tools into the <word-editor> interface above. The change in context of loading an external website, typing a significant amount of content, and then copying and pasting that text back into the <word-editor> is far too slow to be of use during fieldwork, especially when such steps try the patience of the speaker.

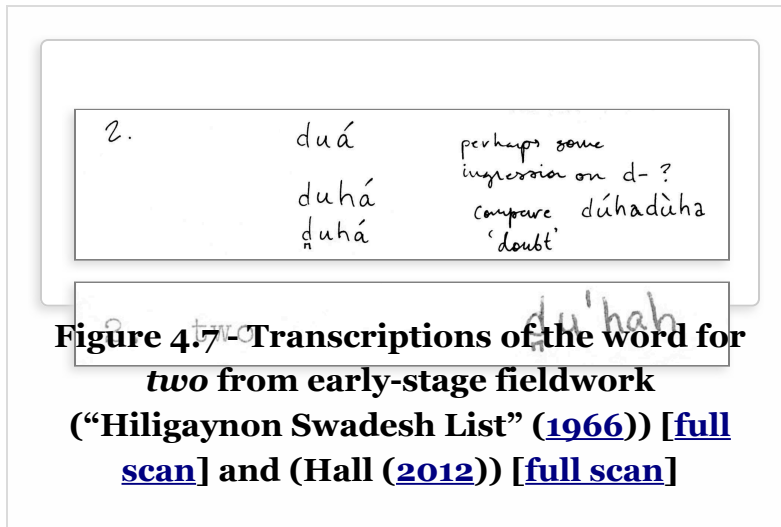
There is another fundamental problem with these interfaces: they do not make the featural phonetic details associated with each of the characters searchable. Linguists are able to make use of the IPA charts because they are trained in phonetics: they understand that a character such as «β», «b» and «B» symbolize particular speech sounds with particular phonetic features (a voiced bilabial

fricative, a voiced bilabial implosive, and a voiced bilabial trill). Understanding that system makes looking up characters in a table possible. That knowledge can be a crucial part of transcription: the IPA charts (and the emulated interactive interfaces above) are practical indices of the IPA alphabet precisely because they are organized in terms of phonetic features: if you know how the features correspond to the charts, you can usually locate the character corresponding to particular phonetic features.

The challenge as far as the working linguist is concerned *during* fieldwork is to convert that phonetic knowledge into characters on the screen. In this section we will explore the design of a transcription component which aims to provide a linkage between phonetic features and phonetic characters which can be used quickly, with a minimum of keystrokes. We shall see that this featural phonetic information is helpful not only for finding characters to use in transcription, but also in many other tasks fieldworkers face: building up phonetic inventories, defining phonemic orthographies and transliteration schemes, and finding minimal pairs.

4.4.1 Manuscript transcription

Consider the two short excerpts from early-stage fieldwork of Hiligaynon. The first is from the first page of this author's 2012 fieldwork carried out at UCSB, and the second is from fieldwork carried out in 1965, now preserved in the UCLA Phonetics Lab archive (the full pages are available at the links in the caption). There is nothing particularly remarkable about these excerpts — each contains transcriptions of the Hiligaynon word **duhá** 'two' — except perhaps that despite the nearly 50-year gap between their production, the actual transcriptions are quite similar.



For simplicity I have chosen a word which had only slight variation in transcriptions between these two fieldwork projects. Here is the full list of transcriptions of this word from the two sources:

1965	ɖu'hah
	duá
2012	duhá
	ɖuhá

Documentary linguists must carry out some mental gymnastics during fieldwork transcription of this kind. Remembering that the specific diacritic which indicates that the voiced plosive “d-sound” has a “dental” place of articulation is the “bridge” [ɖ] is a feat of pure memorization. (It is not to be confused, for instance, with the *inverted* bridge [ɗ] — which specifies apical place of articulation.) One would typically simply look in the IPA diacritic chart, where diacritics are listed together

with associated phonetic or suprasegmental features:

◌̥ centralized	◌̘ mid-centralized	◌̜ lowered
◌̥ḥ	◌̥̄ tie bar	◌̥' ejective
◌̥ṯ dental	◌̥ḥ aspirated	◌̥̚
◌̥̚ velarized or pharyngealized	◌̥̚ linguolabial	◌̥̚ laminal
◌̥̚ apical	◌̥̚ advanced	◌̥̚ retracted
◌̥̚ raised	◌̥̚ advanced tongue root	◌̥̚ retracted tongue root
◌̥̚ less rounded	◌̥̚ more rounded	◌̥̚ voiced
◌̥̚ voiceless	◌̥̚ creaky voiced	◌̥̚ breathy voiced
◌̥̚ syllabic	◌̥̚ primary stress	◌̥̚ secondary stress
◌̥̚ non-syllabic	◌̥̚ nasalized	◌̥̚ extra-short
◌̥̚ no audible release	◌̥̚ lateral release	◌̥̚ nasal release
◌̥̚ labialized	◌̥̚ palatalized	◌̥̚ velarized
◌̥̚ pharyngealized	◌̥̚ long	◌̥̚ half-long

This list is not enormous, and once found, a given character could simply be copied into a manuscript (modulo the difficulties of cutting-and-pasting diacritic marks). But note, crucially, that we have found this character via linguistic labels, not typographical labels. Linguists looking for the diacritic for “dental place” are more likely to be familiar with a term from articulatory phonetics (“dental”) than they are with an obscure typographical label such as “bridge”.¹⁷ So we need to provide a mapping between the phonetic nomenclature which is familiar to linguists on the one hand, and the full repertoire of IPA characters, on the other.


This raises a challenge, however: the range of possible speech sounds that has been documented is in the thousands. The current state of the PHOIBLE database of phonological inventory data contains 3,183 entries ([Moran and McCloy 2019](#)). In a useful overview of the design and history of the International Phonetic Alphabet (IPA) from its origins to the Unicode era, Moran and Cysouw¹⁸ point out a design feature of the IPA which explains how the IPA can serve as a way to encode such complexity. The IPA itself — the alphabet or repertoire of identified characters — is not a database of speech sounds. It is, rather, a featural system: it enumerates symbols that stand for a particular permutations of phonetic features:

The IPA is intended to be a set of symbols for representing all the possible sounds of the world’s languages. The representation of these sounds uses a set of phonetic categories which describe how each sound is made. These categories define a number of natural classes of sounds that operate in phonological rules and historical sound changes. The symbols of the IPA are shorthand ways of indicating certain intersections of these categories.


Roach ([1989](#)), quoted in Moran and Cysouw ([2018](#))

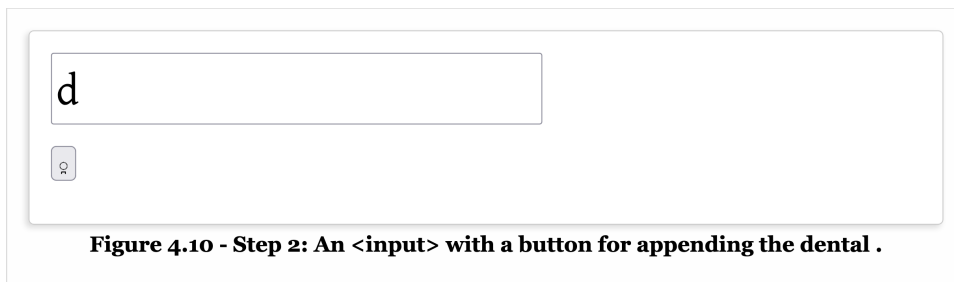
This quote is an explicit statement that the IPA is intended to be understood as encoding phonetic features. For example, the IPA character d (identified by U+0064 in Unicode) is by default associated with the following phonetic features (they may be read directly off the IPA pulmonic consonant chart):

voicing	voiced
place	alveolar
manner	plosive

Likewise, the diacritic mark , known in Unicode as COMBINING BRIDGE BELOW (Unicode codepoint U+032A), is associated in the IPA with just one phonetic feature: the *dental* place of articulation.

place dental

This featural design means that there is no unitary Unicode character corresponding to the “voiced dental stop”, . As far as Unicode is concerned, that representation of a phone is a sequence of the two characters — in a piece of transcribed text, it is equivalent to the sequence U+0064 followed by U+032A. We may represent this featural arithmetic schematically as below:



Linguists already know this information. In learning to use the IPA, they see how its characters can be combined into a symbolic representation of phonetic features and values that they already know and understand. But if we wish to build user interfaces which help linguists *during* fieldwork, then we must be sure that such basic knowledge is “implemented” in a familiar, usable way in the user interface.

4.5.1 Finding characters and phones

Given this background, we can describe the steps which must be carried out during

fieldwork in order to create arbitrary IPA transcriptions:

Figure 4.9 - Phonetic transcription workflow

1. Find necessary phonetic character via phonetic (and not typographical) nomenclature.
2. Insert that character into a text input.

We may begin with the second task: one simple approach to aiding the user in inserting a character is simply to provide a button for each desired character directly adjacent to the input field where it is to be used. In the demo below, clicking the button containing the dental diacritic «◌̪» appends it to the «d» character already present in the input:



Figure 4.10 - Step 2: An <input> with a button for appending the dental .

This is a reasonably intuitive, if imperfect, user interface. We will explore some iterations of a design based on this input to the point that it enables the workflow as defined. All interfaces are imperfect. The reader may wish to consider their own reactions to the utility of this component as we work through its design: what specific changes might improve it?

4.6.1 A user interface design for inputting phonetic transcriptions

Before we proceed with the design of our IPA input interface, we should pause to recall that there is no such thing as a perfect user interface. IPA data is large and complex; there are many possible ways that we could assist a user in inputting IPA transcriptions. The design below may or may not meet the needs of a given documentation project, but it does demonstrate how key Unicode and phonetic data can be integrated into a usable interface.

The IPA defines a repertoire of characters, but it is not an inventory of speech sounds (phones) — although sometimes the two categories overlap: the character «d» *can* represent a specific speech sound: a (pulmonic) voiced alveolar plosive consonant. And that character is part of Unicode. But there is no character corresponding to the voiced *dental* plosive in the IPA. To represent that phone with Unicode, one must use a combination of «d» with «◌̪», as described above. Which characters *can* be combined is a matter of linguistic knowledge, not Unicode encoding. It is also a matter of historical happenstance: an explanation for why the labiodental fricative has its own unitary Unicode character ([f]) but the labiodental plosive does not (it is spelled [p̪]) would have to resort to many details about the

history of the IPA which are not entirely linguistic in nature. (It is probably relevant that many European languages have [f], few have [p̪].)

The interface below allows the user to find Unicode IPA characters via a string of search terms identifying phonetic features. To see how the search works, try clicking the button at the beginning of each list item to see the query run:

1. - search which matches uvular consonants
2. - matches voiced fricatives that are palatal *or* palato-alveolar (search term order is arbitrary)
3. - search which matches uvular plosive consonants
4. - matches the aspiration modifier letter [ʰ]
5. - abbreviated version of previous
6. - search which matches front rounded vowels

You may also of course type similar queries directly in the filter box. A button is generated for each matching phone, but as of yet they do nothing — the character palette is managed by this component, but it must be embedded into another component to be used.

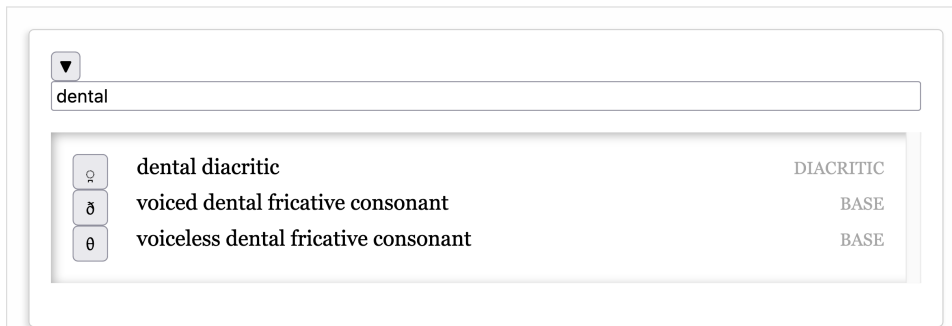


Figure 4.11 - <ipa-palette>: A text palette with added component for finding and inserting Unicode IPA characters via phonetic features

At last we are in a position to create a useful text input interface for arbitrary IPA text. To do this, we will embed our <ipa-palette> component inside a new component, <ipa-input>. This latter component provides a (large) text field for inputting text directly, but also hooks up the buttons in the <ipa-palette> to insert characters wherever the cursor is in that field.

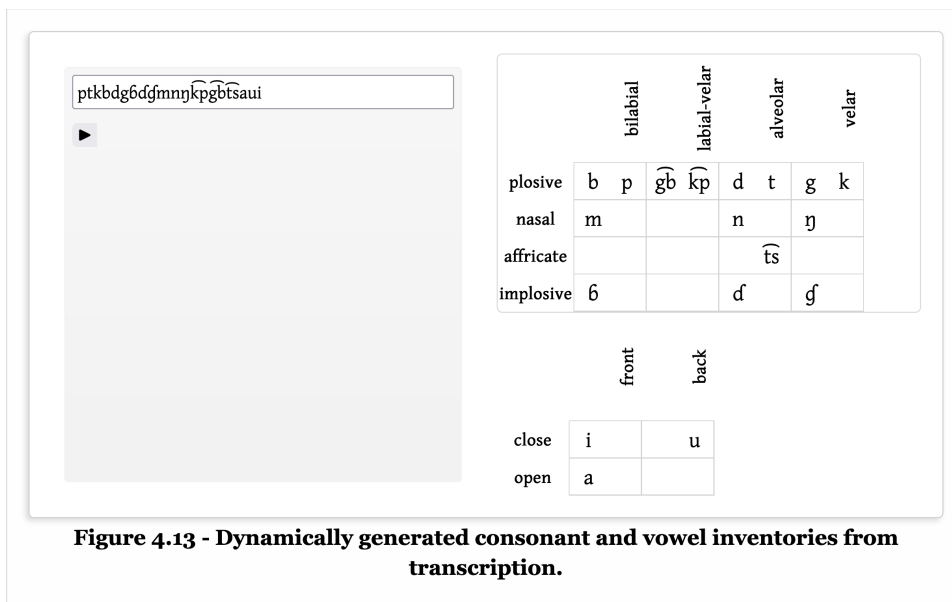


Figure 4.13 - Dynamically generated consonant and vowel inventories from transcription.

4.7.1 Inferring phonetic inventories from transcriptions

Because <ipa-input> associates phonetic features with Unicode characters, and because Unicode defines whether individual characters are “base” characters, diacritics, or modifiers (see Davis and Heninger ([2020](#))), it is possible to derive a basic inventory of phones directly from transcriptions. This functionality is demonstrated below, where an IPA input automatically updates consonant and vowel inventory charts as the user transcribes. Try pasting in some random characters such as ptkbdg6dgmnykpgbtsau into the input to see the generated inventory.

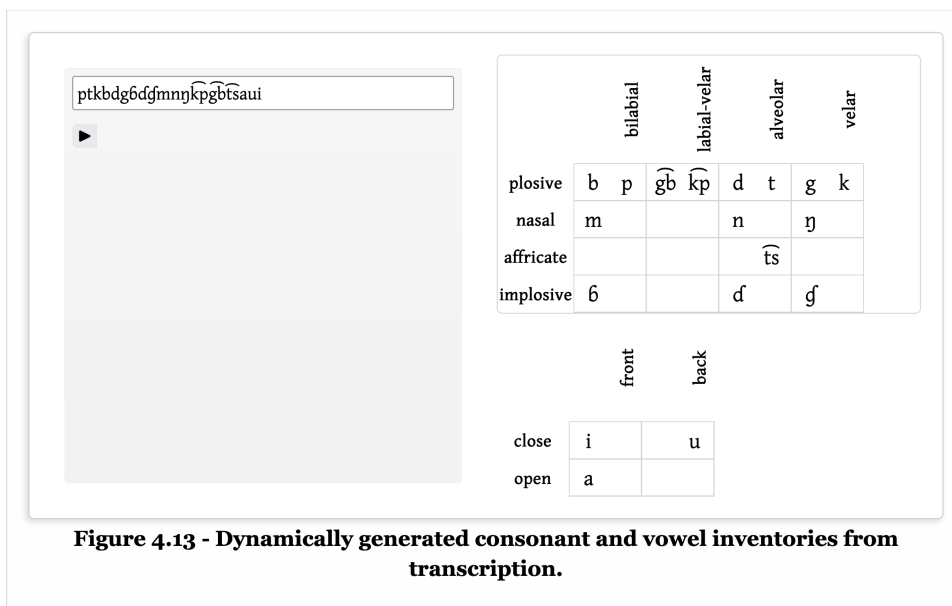


Figure 4.13 - Dynamically generated consonant and vowel inventories from transcription.

This interface recognizes the difference between “base” characters such as «p» and «θ» and “modifier” letters such as «i» or «h», thus can classify a phone such as «p^h» as a unitary grapheme. Based on the values for the place, manner, and voicing features of each such phone, the consonant chart is generated dynamically from the

transcription. This makes the chart available in concert with the process of transcription, an interesting variation on the way such charts are often treated during fieldwork: as outputs, rather than tools which are available for reference throughout the documentation process. We shall see below that the same principle can be put to work at a larger scale than a single, one-line input field.

In the next section, we will further explore ways in which the context of fieldwork influences the design of software for carrying out fieldwork.

4.8 From workflow to interface

In this section we investigate the relationships between two very different workflows for the same dataflow. In the first, we consider a traditional user interface for eliciting the Hiligaynon Swadesh list described above, one which is very similar in layout to the paper elicitation schedule. In the second, we consider a novel interface where textual prompts are replaced by images.

4.9.1 Scheduled vocabulary

In the current example we will consider how the `<ipa-input>` component described above can be incorporated into a very traditional user interface for eliciting a “scheduled” vocabulary.

The scanned document below is the complete record of the resource described above ([“Hiligaynon Swadesh List” 1966](#)), a “Swadesh list”¹⁹ elicitation carried out by

the UCLA Phonology Lab. This particular resource is interesting because it allows us to reconstruct the specifics of the workflow by comparing the recording and the scanned transcription.

English	Hiligaynon	English	Hiligaynon
1. one	i'sah	26. tooth	'hi'pun
2. two	du'hah	27. tongue	'di'la?
3. big	da'ke?	28. fingernail	ka'ke?
4. long	ma'la ba?	29. foot	'ti'zil
5. small	'di'fah	30. knee	'tu'hu?
6. woman	ba'bayeh	31. hand	ka'mat?
7. man	la'lakeh	32. neck	'li'g?
8. fish	'ie'qa?	33. heart	ku'a'sa'n
9. bird	'pi'spe's	34. sun	'ad'la?
10. dog	i'ge?	35. moon	'bu'lan
11. tree	'ka'ho?	36. star	bi'tu'uh
12. skin	'pa'ni't	37. water	'tu'bi?
13. flesh	u'ha?	38. rain	u'lan
14. blood	du'ge?	39. stone	ba'ta'h
15. bone	'tu'la'?	40. sand	ba'las
16. egg	'i'te'la'g?	41. earth	'du'ta?
17. horn	'su'gai	42. smoke	a'sa
18. tail	'i'ka'g?	43. fire	ka'la'yo
19. feather	'bu'l bul	44. mountain	'bu'ki?
20. hair	bu'ha?	45. night	'ga'be'h
21. head	'u'la	46. hot	ma'ri'ni't
22. ear	du'lungan	47. cold	ma'tu'ga?
23. eye	ma'tah	48. full	pu'ne?
24. nose	i'le'g	49. good	ma'zi'ye
25. mouth	'ba'ba?	50. round	ma'ti'pu'lon

Figure 4.15 - Full page of Hiligaynon elicitation sheet from UCLA phonetics archive

Although the transcript is “analog”, the physical paper itself did serve as a kind of user interface for the participants. It contained both readable glosses to serve as prompts, and as a medium on which to record transcriptions of corresponding forms. But careful reflection on how workflow and output intertwined in this project can help us to better understand both how to implement a digital user interface, and

to foreground opportunities for the integration of aids which stand to make the process more efficient and the resulting data more useful.



**Figure 4.16 - Sample word from UCLA
Hiligaynon elicitation schedule, mə'ʔinɪt' 'hot'**

Recalling that this fieldwork was carried out under the auspices of the UCLA Phonology lab, it is not surprising that what is transcribed is a close phonetic representation without morphological analysis. ²⁰

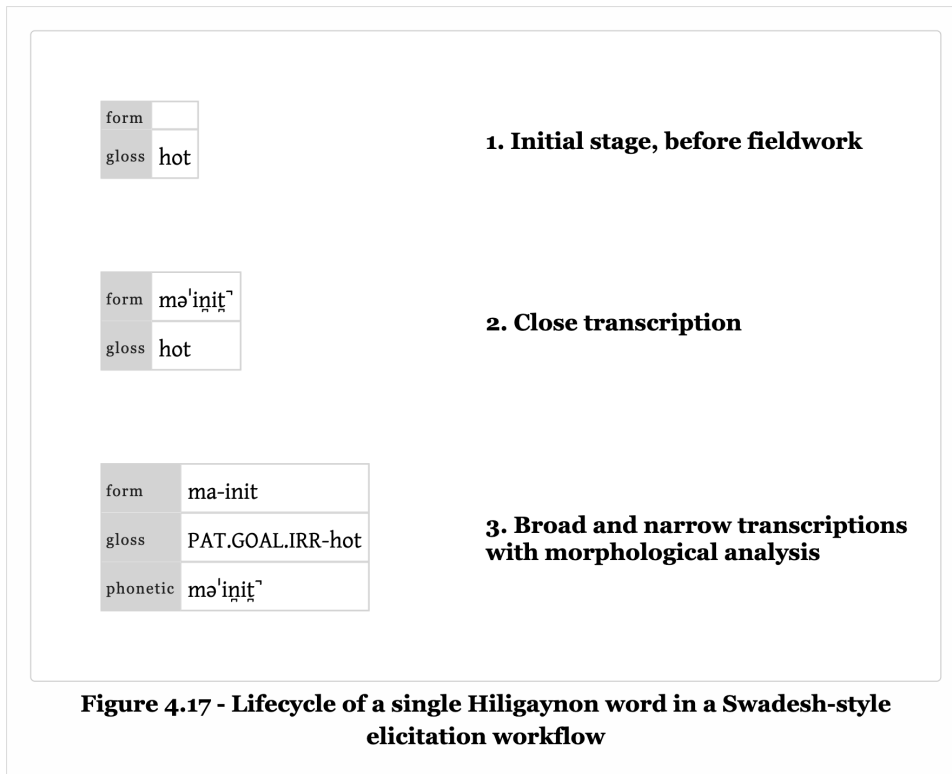
form	mə'ʔinɪt'
gloss	hot
metadata	number 46

However, a documentary linguist would be likely to revise this representation, given that the word has internal morphological structure — the prefix transcribed (narrowly) here as *mə* is one of a complex set of verbal prefixes in Hiligaynon. Morphological glossing might be captured using the rules described in the Leipzig Glossing Rules ([Comrie, Haspelmath, and Bickel 2015](#)). A broader orthography might also be employed.²¹

form	ma-init
gloss	PAT.GOAL.IRR-hot

The Hiligaynon word *mainit* with Leipzig-glossing-style gloss.

Note, then, that we can think of this data structure as “evolving” in keeping with research aims. The diagram below portrays this as a “lifecycle”, summarizing how the attributes on a documentary data object might change in accordance with research goals. Again, it should be emphasized that as long as the core “identifying” attributes of a particular data type are present (in the case of `Word` objects, something corresponding to the `form` and something to the `gloss`), then it is natural and expected that other attributes might be added. Any component which can render a default `Word` object will be able to render that object even if not all of its attributes are being displayed.



Although we have thus far been using a “working model” of a `Word` data type which was defined as the set of `form` and `gloss` properties, this particular documentary workflow involves progressive modification to the data structure, even re-casting the narrow transcription mə'ɪnɪt̪ʰ from being described as the “basic” `form` to being a `phonetic` transcription. There is nothing atypical about modifying data progressively in this way. Data structures are to be defined by their uses: to the phonologist, the structure seen in “step 1” might serve as an endpoint (and in fact, the sample data was archived in that form). Just so, a historical linguist might recombine these values and properties in yet another way, using a very terse `gloss` such as `HOT` with a broadly transcribed phonemic `form` (perhaps *mainit*) for the purposes of historical reconstruction. Our purpose here is not to evaluate such transcriptions, but simply to stress that the shape of data structures is strongly

influenced by the nature of research goals and workflows. Consequently, we should expect that user interfaces designed to support differing workflows — even when those workflows result in an identical data transformation — will have quite different user interface controls and functionality.

An aside on representing grammatical categories

I have skirted somewhat the issue of the detailed representation of how morphological glossing and grammatical category labels should be handled. A full account of data modeling for grammatical classification is beyond the scope of the current work, but we can take a brief look here at how a simple “grammar object” may be structured, in keeping with the notion of the “Boasian Database”.

metadata	name	Hiligaynon Grammatical categories	
	notes	Analysis based on Wolfenden (1971) ‘Hiligaynon Reference Grammar’.	
categories	category	value	symbol
	trigger	patient	PAT
	focus	goal	GOAL
	aspect	irrealis	IRR
	mode	neutral	NEUT

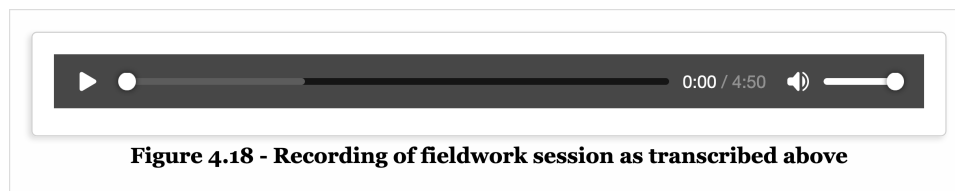
The `symbol` field is the familiar abbreviation for the grammatical category label; by associating traditional labels with both the category name *and* the value (or exponent) of that category, we maintain the link between familiar notation (PAT, IRR, etc.) and the structured data which represents the category-value relationship within the grammatical system. This analysis was chosen precisely because it is

rather idiosyncratic; whatever one's evaluation of Wolfenden's categorization of verbal affixes into categories such as `trigger` might be, this data structure remains a useful way to *capture* such classifications.

There is much more to be said about the representation and use of grammatical data such as this, but I would like to point out an important fact: because this structure links abbreviations (“symbols”), categories, and values, modern Leipzig-style morphological glossing can be treated as a kind of serialized database, e.g.: because PAT is stored as the value “patient” of the category “trigger”, we could then retrieve any verb via either value or category. That is to say, “Show me all verbs which are marked with a value for ‘trigger’”, becomes as feasible as “Show me all verbs glossed with the PAT abbreviation.” Such flexibility is very important, as it enables us to search not only for values of particular grammatical categories, but also *the categories themselves*.

See [§5.2.3](#) for some thoughts on possible further development of this approach.

It is instructive to listen to the first minute or so of the Swadesh list recording, as recorded here:



The way that the recitation occurred (together with the scanned Swadesh list) makes it clear that the prompts were pre-arranged or “scheduled”.

Participant A: *I come from the Philippines, particularly in La Paz, Iloilo City. My native language is Hiligaynon.*

Participant B: Following are 50 words from the Swadesh Diagnostic word list.

Participant A: ‘One’, *i 'sah, i 'sah, i 'sah.*

Participant A: ‘Two’, *ɖu 'hah, ɖu 'hah, ɖu 'hah.*

Participant A: ‘Three’, *ɖə 'koəʔ, ɖə 'koəʔ, ɖə 'koəʔ.*

It is likely that there were three separate workflows involved in this fieldwork.

Workflow 1: Create (typewritten) Swadesh list elicitation schedule (Participant B?)

Workflow 2: Record recitation (Participant A)

Workflow 3: Transcribe the recording (Participant B? Some third participant?)

We can infer that this is what happened because the recording of the audio is too fluent to have allowed for the simultaneous narrow phonetic transcription — the transcriptions must have been carried out after the whole recording was complete. Note also that strictly speaking, those transcriptions do not represent the entirety of what was spoken: the speaker uttered each Hiligaynon word three times, but there was just one (manuscript) transcription per word in the document. Below is an implementation of a simple `<lexicon-editor>` (with transcriptions pre-populated) which recreates the original “user interface” fairly closely. Note that this interface incorporates the `<ipa-input>` component described above. In principle, it might be possible to use that capability to enable transcription in a close

orthography that would be fast enough that the linguist and speaker could alternate recording and transcription.

Note also that the design of this user interface recreates the “by-column” layout of the original document, which is actually less than ideal in the “scrolled” environment of a web browser, since upon completion of word #25, the user must scroll all the way back up to continue with #26. CSS allows us to change this default to number the words across rows. Compare the two layouts by clicking here:

[toggle direction](#)

Figure 4.19 · A recreation of the original Swadesh list typescript as a user interface

1. one	/səh	26. tooth	/ypu
2. two	dəʔəh	27. tongue	/laʔ
3. big	dəʔəʔ	28. fingernail	/əʔə
4. long	məʔəʔ	29. foot	/ʔi
5. small	/əʔəʔ	30. knee	/əʔəʔ
6. woman	bəʔəʔə	31. hand	kəʔəʔ
7. man	bəʔəʔə	32. neck	/ʔəʔ
8. fish	/əʔəʔ	33. heart	kəʔəʔə
9. bird	/əʔəʔə	34. sun	əʔəʔ
10. dog	/əʔəʔ	35. moon	/əʔə
11. tree	/əʔəʔ	36. star	bəʔəʔə
12. skin	/əʔəʔ	37. water	/əʔəʔ
13. flesh	/əʔəʔ	38. rain	/əʔə
14. blood	dəʔəʔəʔ	39. stone	bəʔəʔ
15. bone	/əʔəʔə	40. sand	bəʔəʔ
16. egg	/əʔəʔ	41. earth	/əʔəʔ
17. horn	/əʔəʔ	42. smoke	/əʔə
18. tail	/əʔəʔ	43. fire	kəʔəʔə
19. feather	/əʔəʔə	44. mountain	/əʔəʔ
20. hair	bəʔəʔ	45. night	/əʔəʔə
21. head	/əʔə	46. hot	məʔəʔə
22. ear	dəʔəʔəʔ	47. cold	məʔəʔə
23. eye	məʔəʔ	48. full	pəʔəʔ
24. nose	/əʔə	49. good	məʔəʔə
25. mouth	bəʔəʔ	50. round	məʔəʔə

As always, there is room for improvement in this interface. Most glaringly, it is a poor workflow to have to continually re-select individual phonetic characters via the <ipa-palette> when editing each form — it would be much more convenient to make the characters chosen with the character palette available across all the form editors. Making this change would be a good next iteration in the development of this interface. Another improvement would be to decide on how best to handle inputting a broad transcription in addition to the narrow transcription.

Design is iterative

In this author’s experience, it is often the case that what constitutes a “good” user interface only becomes apparent after creating a usable prototype. It would seem obvious that adding the <ipa-input> component inside a lexicon-editing interface would make it easier to input IPA transcriptions — and it does, to an extent. But in the context of editing 50 words, it immediately becomes clear that the <ipa-input>s should share “state” — if a phonetic character is useful in transcribing one form, of course, it is likely to come up again. Designing user interfaces is always an iterative process.

4.10.1 Image prompts: sharing an interface between both participants

In the previous interface design, we discussed the design of a user interface which was quite similar to its traditional analog “user interface”: a Swadesh elicitation “schedule”. We thought about the independent workflows which constituted the whole documentation task, noting that the transcription stage was likely carried out by the linguist after the speaker’s work was completed. Consequently, both the original schedule and the modern user interface are oriented toward the participant that is carrying out the transcription role — the basic workflow here is not that different from using a standard spreadsheet interface and filling out forms and glosses in a row-by-row fashion (aside, of course, from the input of a complex phonetic orthography). In such a workflow, and in cases where the participant who is speaking is not the same as the one doing the transcription, then it is the transcriber who is most likely to be “using” the user interface, in the sense of actually carrying out editing of transcriptions.

But what if we questioned this assumption? There are, of course, good reasons to avoid direct translation from the working language — in this case, English. Let us consider a different stage of documentation, one where the phonetic inventory has stabilized and a phonemic orthography is in place. At this stage the focus shifts from phonetic nuance to other levels of analysis: morphology, syntax, lexis, and so forth. A common task in this stage is to collect vocabulary within a particular semantic domain — we'll consider the names of various fruits.

A very simple modification to an interface like the previous can enable quite different workflows: by simply replacing the English `gloss` with an image, the user interface becomes one that both the linguist and the speaker use together. By styling the interface with some dramatic images and typography, the tone of the workflow can change quite a bit as well. Tedium is a very real concern when working in fieldwork. Putting work ahead of time into creating a user interface that has some aesthetic appeal can express respect and appreciation for the speaker's contributions, and can make the task more interesting and fun for all involved. (Furthermore, interesting images may stimulate other genres of fieldwork: short narratives about memories related to the image. In this author's experience working with Hiligaynon, the image of a starfruit prompted a story about how ubiquitous and inexpensive they are in the Philippines, but are considered gourmet and expensive in the United States.)

In the interface below, only the image and a form transcription input is visible. The English “gloss” is present in the data structure as it was before, but only the image is shown. The images were chosen for an unambiguous view of the fruit, but

also because they are pleasing to look at. (Images: su-lin (2007), Katigbak (2009), H (2016), S (2016), Bharatan (2009).)

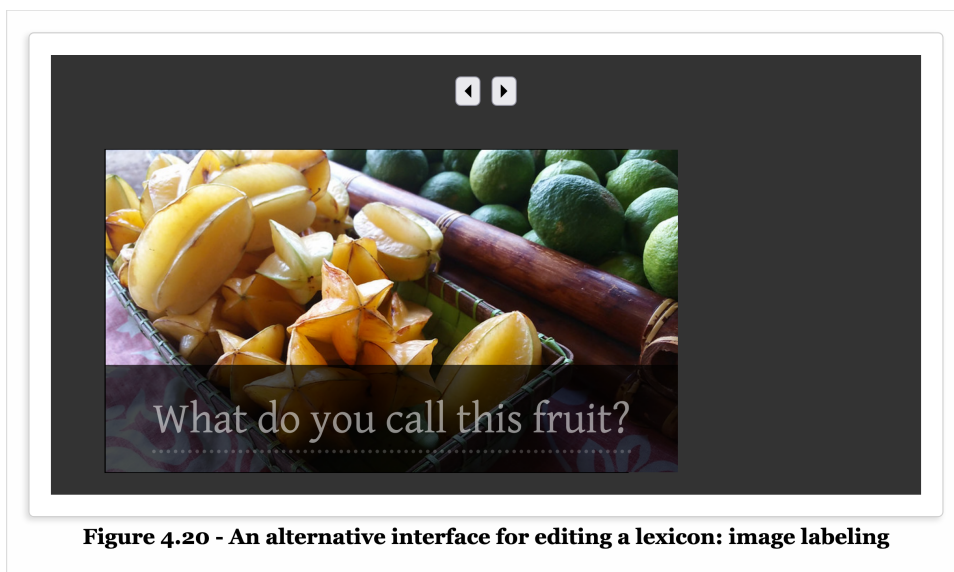


Figure 4.20 - An alternative interface for editing a lexicon: image labeling

Note also that this interface is *paginated*: it looks more like a photo album than an endless list, contributing a small element of surprise and interest. A change as simple as this can improve the speaker's experience of the documentation task. Note that the inline demonstration above can also be presented in an immersive, full-screen user interface. To try the fullscreen experience, [Click here](#) (hit Escape to return).

4.1 Harmonizing time-aligned data

Ultimately, our goal in creating digital fieldwork is to increase our ability to recall all instances of a given linguistic unit, such that we can better evaluate its behavior in all known contexts. In this section we consider the design of an interface which adds

audio capabilities to the word elicitation interfaces we just discussed.

We noted an interesting discrepancy between the fieldwork recording and the contents of the transcription in the scheduled vocabulary user interface. In the original recording, each word was uttered three times, but the transcription summarized those three utterances with a single transcription. Thus, the output data of that workflow was not “time-aligned” in that documentation project: the three utterances are treated as a small database of evidence for an idealized (or “phonemecized”) transcription of the form. (It is of course possible to easily recover the utterances of a given word within the recording by a simple search, but it remains true that no utterance-level timestamps were recorded.)

As we mentioned in [Chapter 2](#), Musgrave and Thieberger (2012) suggest that we think of media as a fourth pillar of the Boasian trilogy. But from the point of view of developing a Boasian database, it seems better to see time-alignment of media as being closely linked to the corpus, not to the lexicon or grammar. In that approach, linkages within the dictionary and grammar become references to the corpus.

Let us consider what design changes would be required in order to add audio recording capabilities to our image labeling interface. As always, there are three main questions to answer:

1. What is the dataflow?
2. What is the workflow?
3. What is the user interface?

It is straightforward to tabulate what the time-alignment data for the Hiligaynon Swadesh list recording would look like. For instance, for the segment of the recording corresponding to the word **isá** ‘one’, we have a recitation of the English gloss, and three repetitions of the Hiligaynon form (to hear the relevant segment, [Click here](#)). Aside from a slight falling intonation (list intonation) in the final repetition, the utterances are quite similar. Thus the only difference in each time-aligned transcription are the `start` and `end` times.

transcription	translation	language	start	end
one	—	English	17.24	17.96
isá	one	Hiligaynon	19.4	20.15
isá	one	Hiligaynon	20.6	21.37
isá	one	Hiligaynon	21.65	22.25

Note also, however, that we have changed the field names in this data. Rather than using `form` and `gloss`, we have switched to using `transcription` and `translation`. This is because we observe the convention that *only texts have time-alignment information as primary data*. Word objects within a `Lexicon` should only maintain time-alignment information via reference to `Sentence` or `Word` instances in a specific `Text`. This puts us in a better position to reconsider the time-alignment data produced in our first discussion of the iterative recording interface from Chapter 1. By treating the iterative recording interface as producing `Text` data rather than `Lexicon` data, we are actually in a better position to *merge* data from distinct sources, thus increasing the accountability of our documentation. For instance, the utterances of the common word *isá* ‘one’ from the UCLA recording could be easily merged with content from some other long fieldwork recording which has been captured as a `Text JSON` structure.

*I note here in passing that I only came to this conclusion about the granularity of timestamps being germane to `Texts` rather than `Lexicons` after I had implemented the iterative recording interface in Chapter 1. In hindsight, it makes much more sense to think of that user interface as creating `Sentence` data rather than `Word` data, but it did not seem so obvious as the first interface was being created — it seemed obvious that we were “recording words”. Herein lies a lesson: sometimes the best way to better understand how to structure our data comes from attempting to build a user interface to **do** something with that data.*

Let us assume, then, that the time-alignment information tabulated above will be produced interactively, and that they will be captured within an instance of the `Text` data structure, with each utterance corresponding to a `Sentence` rather than a `Word`. We can then programmatically derive a `Lexicon` which references each time-aligned utterance.

One of the appealing features of a component-based approach is that it is possible to create interfaces from existing components. In this case, we can reuse (with slight modifications) the audio recording components that were used in the iterative recording interface described in Chapter 1. An excellent way to begin the design process is with a rough sketch such as the one below:

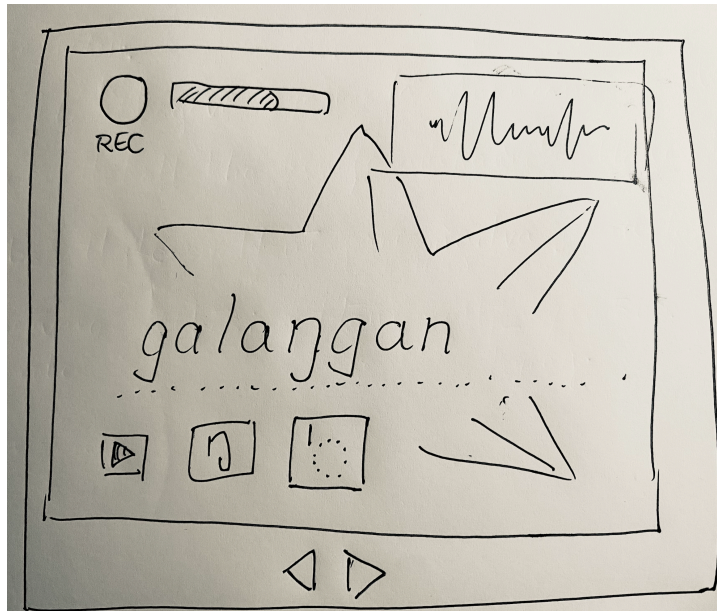


Figure 4.21 - A sketch design for an interface that combined image-labeling, audio-recording, and IPA input.

Below, the sketch design is labeled to show how it is built up from other components:

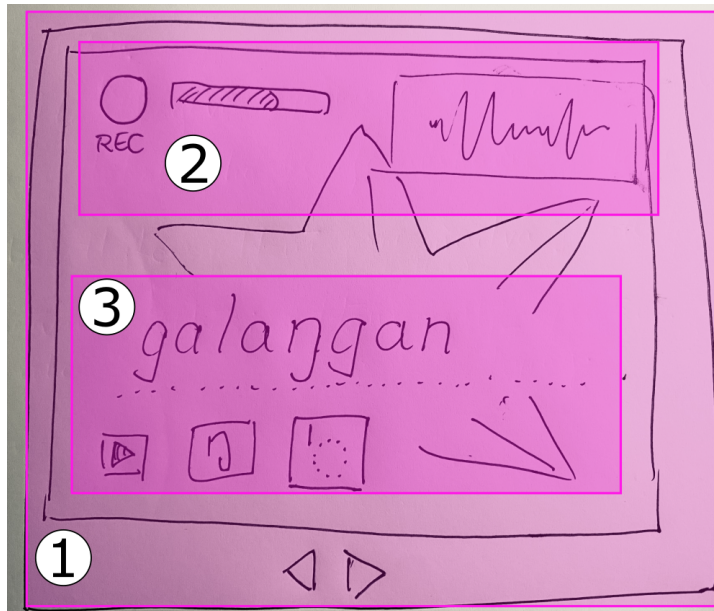


Figure 4.22 - The constituent components in the previous design: (1) the image labeler component; (2) the audio-recorder component; and (3) an IPA input component.

While we have not completely implemented this design, a diagram like this is an excellent (and necessary) step towards implementation. A design at this level of granularity can serve as a clear roadmap for implementing a working user interface relatively quickly: the individual component parts are already working, and the input and output data are well defined. Like the previous image labeling interface, this interface offers opportunities to modulate the relationship between the linguist and the speaker during fieldwork.

The reader is encouraged to reflect on their own reaction to the example designs above: would any of these have met your needs in your own fieldwork experience? For what reasons? If not, what workflow steps are not addressed in these designs? Analyzing dataflows and workflows precisely is a skill which can be learned and practiced. That skill, in turn, better enables linguists to participate in the design of new user interfaces which might actually be implemented — after all, there can be no implementation of software without a clear statement of exactly what the software is meant to do, and how it should work.

4.1 Automatic data updates depend on User Interface Design

While the data model we have developed here is fairly complete, one area where the current implementation of `docling.js` needs significant development is the degree to which documentary data of various kinds can be automatically synchronized. For instance, the task of “glossing” — morphological annotation of textual materials — for example, is one of the most time-consuming tasks in producing a digital corpus. (Automating the assignment of glosses to forms is the major function of Fieldworks (FLEx).) Such automated synchronization of the content of a corpus and a lexicon, therefore, is of great importance to documentation, and adding user interfaces with this capability to `docling.js` is an important near-term goal.

That it is useful for lexicons and corpora to be automatically synchronized is *prima facie* an obvious desideratum. But how, exactly, is that that synchronization to be maintained with regard to user interface design? For instance, how should the

interface inform the user that the lexicon has been updated? Should a change to a given word be silently propagated across all texts in the whole corpus, or should some process of double-checking each occurrence of a change be put in place whereby the linguist can evaluate the generality of the change? There simply are no generic answers to such questions. The way that data changes as it is created and modified within a user interface is strongly dependent on the kind of workflow the interface is designed to enable.

It is hoped that the examples in the present chapter at least demonstrate the utility of building up composite applications (as we did when we inserted the `<ipa-input>` component into the scheduled vocabulary elicitation interface). Indeed, even that composition has lingering user interface details that could be improved upon: for instance, the state of the button palette should be shared across the whole interface, the drop-down interface is causing the remainder of the content to be pushed down in a confusing way, and so forth. Such problems are not marginal in software design. They are expected to arise as user interfaces become more complex. The `docling.js` library can serve as a starting point for ideating and then prototyping novel interfaces, which can then be improved through testing and critique. It is only through such an iterative, collaborative process that we can begin to solve the larger problem of dynamic linking and harmonization of composite data structures.

Because of these constraints, in the final chapter of this work we will turn to what amounts to social factors in software design for language documentation: building on the existing expertise of linguists will require a kind of design process which

includes as many participants as possible. We will refer to this process as participatory design.

5. Avenues ahead

In this work, we have considered domains that are not traditionally associated with linguistics at all, including data modeling and user interface design. Our goal has been to demonstrate that documentary data may be structured in such a way that it is at once familiar to linguists and amenable to implementation in functioning user interfaces. A simple data model was described — built with the simple notions of *objects* (bundles of property/value pairs) and *arrays* (lists of objects) — which can be used to capture a wide swathe of documentary data types and their interrelationships. In the previous two chapters, we considered the complementary topic of how to design user interfaces for displaying and using such structured data. The details of the workflows that we carry out as we complete various tasks in documentation were shown to warrant close analysis, as it is the details of those steps and the contexts in which they are carried out which constrain the design of appropriate and helpful user interfaces. In this final chapter, we consider how `docling.js` fits into the larger context of the Web Platform, the powerful browser-based computing environment which has evolved from the World Wide Web (which is itself a specific application of more general internet technology). We will also sketch out some suggested trajectories for those who would like to participate in the design of new software interfaces for language documentation.

5.1 The Web Platform and `docling.js`

The Web Platform, a computing environment which can be thought of as “residing” primarily within web browsers, has evolved from the familiar World Wide Web. And yet, the Web Platform as such is unknown to most users of the web. Web browsers are themselves designed to be user-friendly façades over the complexities of the internet itself. Understanding how the familiar web browser application packages and exchanges information via the internet can help us to understand better how our work in user interface design fits into the Web Platform as a whole.

The browser’s user interface is akin to many other “desktop” applications, with the distinction that it retrieves and sends data to and from “websites”, rather than opening and saving them on the same computer that is running the browser. This basic conceptualization of how the World Wide Web functions is roughly correct: web browsers *are* like any other desktop application, except that “files” are replaced with “web pages”, and file names are replaced with *URLs*, Universal Resource Locators (which begin with `http://` for *Hypertext Transfer Protocol*^{[22](#)}). In effect, URLs identify content on the web, and the primary function of the World Wide Web, is to enable document-linking via links within those files. HTTP is the set of rules (or *protocol*) which describes how the web browser (the *client*), and a *server* (a remote computer connected to the internet) exchange data. In the case of the World Wide Web, this data consists of *hypertext* encoded as HTML files. Thus, the client (web browser) sends a *request* to a URL, and the server *responds* with the corresponding page.

But in fact, this data-transfer system is more similar to the “desktop” model — opening a file stored on the same computer as the application is running — than the browser user may realize. It is easy to think of a web page as somehow residing “on” the server — and indeed, it is *stored* on that server — but when we speak of a server “serving” a page to the web browser, what actually happens is that an HTML file is simply *copied* from the server to the user’s computer — what appears to be a “page” is just an HTML file. In fact, it is possible (if unusual) to simply open an HTML file in the same way one might open a word processing file in a word processing program or a .pdf in a PDF viewer (all browser applications’ menus include the same File > Open option as those other programs).

The user may also fill out some information to be processed by a program running on the server. Such a program running on a server is said to be running on the “server side” of the interaction: it “listens” for requests for particular URLs, and finds the corresponding resource, usually an HTML file. Or, if the user sent along information that they input into their browser, the server will run a program to process the information, and perhaps programmatically generate a new HTML page from the output of that program. So for instance, perhaps the user is using a web-based dictionary, the web page containing a simple form with an `<input>` for a form, an `<input>` for a gloss, and a `<button>` to submit the word might allow them to send a request to `https://example.com/dictionary/add-word`, and “submit” a new word at that URL, perhaps the form *gato* and the gloss *cat*. The server in turn “decodes” that encoded information, and runs a program which adds it to its database. When the submission is complete, the server will generate a response page that contains text like “thanks for contributing the word *gato* ‘cat!’”. Now the

database has been updated, and other users can make use of it. One obvious use would be search: a user might fill out a form at `https://example.com/dictionary/search/` which contains a field to enter an unknown form. If they fill in that input with “*gato*”, the server will receive the query, run its search program over the database, and finally generate a response page displaying the form with its gloss. For most of the history of the web, these two basic kinds of requests (i.e., those for HTML files “at” a particular URL and requests that pass information to a server-side program) encompassed the vast majority of user interactions.

However, a subtle shift has taken place: the *web* — still very much in use as just described — has *also* evolved into the “Web Platform”. To understand what this means, consider which of the two computers involved in requests and responses — the user’s computer acting as a “client” and the internet computer running a website and thus acting as a “server” — is actually carrying out more computation.

In the scenario described above, all of the database-related functionality is carried out on the server: adding terms, running queries, and dynamically generating pages of results. When we talk about the “Web Platform”, we’re talking about functionality that is *not* carried out by a remote server, but which is instead carried out on the user’s computer, by the browser application itself. The user interfaces we have been describing here are, in effect, applications which are themselves running “on” the browser application. It is this relationship which explains the term “platform”: it is called the “Web Platform” because it arose within the context of the web, but in fact a more easily understandable name might have been the “Browser Platform”: the browser as a computing and programming environment in its own

right. The crucial feature of this programming environment is that it has its own “native” programming language, known as Javascript. The browser knows how to execute computer programs without any assistance from the server which are embedded *inside* of a `<script>` tag inside of an HTML page²³.

This is a powerful idea, because it means that web “pages” which are delivered in the same way as a traditional HTML document can also function as an interactive application. To return to the dictionary example, rather than divvying up the work such that the browser’s only job is to send off word submissions and display results generated on the server, the server might send an HTML file called `dictionary.html` which contains all of the embedded programs necessary to run a complete dictionary application with no further interaction with the server at all. Inside `dictionary.html` there would be a `<script>` tag containing a Javascript program that contains the entire dictionary, plus code to implement “search” routines, render results, and so forth. In this usage pattern, the server’s only role is to provide the HTML file with its embedded Javascript code.

But beyond this basic means of providing interactivity, the Web Platform consists of a whole suite of additional functionalities which are now standardized, programmable functionalities of web browsers — each of which can be used from within Javascript programs, and thus made interactive as well. These individual functionalities are called (again, a sensible acronym) APIs, for “Application Programming Interfaces”. One might think of Web APIs as the list of all the things modern browsers can do, and the range of functionalities, and thus, the possibilities for designing and implementing applications, is quite extensive. As of 2021, there are

almost 100 different Web APIs. ([Hegaret 2011](#)) We have already made use of some of these APIs “behind the scenes” in the examples in this work, including the Media Stream Recording API, which is an interface for recording audio and video in the browser; the Custom Elements API, for developing Web Components, and many more. One of the unique aspects of the Web Platform is that all of these APIs are available for development *in a single computing environment*, the browser itself. Any of these functionalities may be combined by Web Platform applications. The Iterative Recording interface we saw in chapter 1, for instance, is only possible because the Media Stream Recording API is part of the Web Platform. The Web Platform enables and encourages the combination of functionalities in novel ways.

Furthermore, although we have just been at pains to distinguish between the (earliest) server-oriented original pattern and the modern browser-oriented pattern, these two approaches are in fact two sides of the same coin: an application which is largely “client-side” in that the majority of necessary computation is carried out by Javascript programs running within the browser may *also* make use of occasional (or even frequent) requests back to a server (or even multiple servers). Our client-side dictionary application, for instance, might periodically request data updates from a server, even as the user is working with the application. Similarly, the program might allow the user to periodically submit updates back to the server, in a “syncing” pattern.

`docling.js` addresses only the “browser-side” part of this picture however. It is designed to demonstrate how it is possible to build complete user interfaces for documentation within the Web Platform, independently of any server configuration

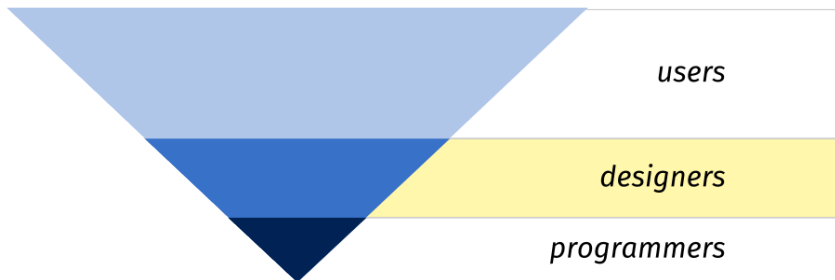
beyond a very basic HTTP server. The existence of the Web Platform means that we may begin the process of designing interfaces to meet our data and workflow needs immediately, *without* committing to any particular server setup. While this might seem like a technical detail, it is an important one, because configuring servers is the least standardized part of the internet technology landscape. Running servers with custom server-side code requires significant investment to reimburse programmers to maintain that specialized server side code. But the basic HTTP server functionality described above is very inexpensive. There are innumerable free or nearly-free options for hosting the HTML, Javascript, CSS, and audio (or even video) files that can be used to create a platform that relies on visitors' computers to carry out the lion's share of the computation. There is a clear path, then, from fostering community engagement in user interface design for documentation interfaces to deploying (where appropriate) those interfaces to the web.

If we predicate the design of new user interfaces on creating general server-side solutions, then we face a different problem, and one which is less amenable to participation by the documentation community at large. User interfaces — the tools we use on a daily basis — are where our expertise as linguists can be most profitably and immediately applied. In the next section I lay out several kinds of participation that interested linguists can pursue in accordance with their own interests.

5.2 Fostering a documentation interface design community

As we saw in the previous two chapters, many variables interact in the design of a

user interface. Fortunately, in language documentation we have a fairly well-established model of documentary data which we can use as a starting point. The opportunity at hand in the present work is to empower linguists with the skills and conceptual framework necessary to participate in a process that applies their specialized working knowledge to bear as a key component of user interface design space.



The diagram above portrays a way of thinking about how a community of participation for building documentation tools could take shape. The triangle classifies roles that participants in a digital documentation project might choose to assume, and the size of each layer corresponds to the “slice” of the documentation community that might participate at that layer.

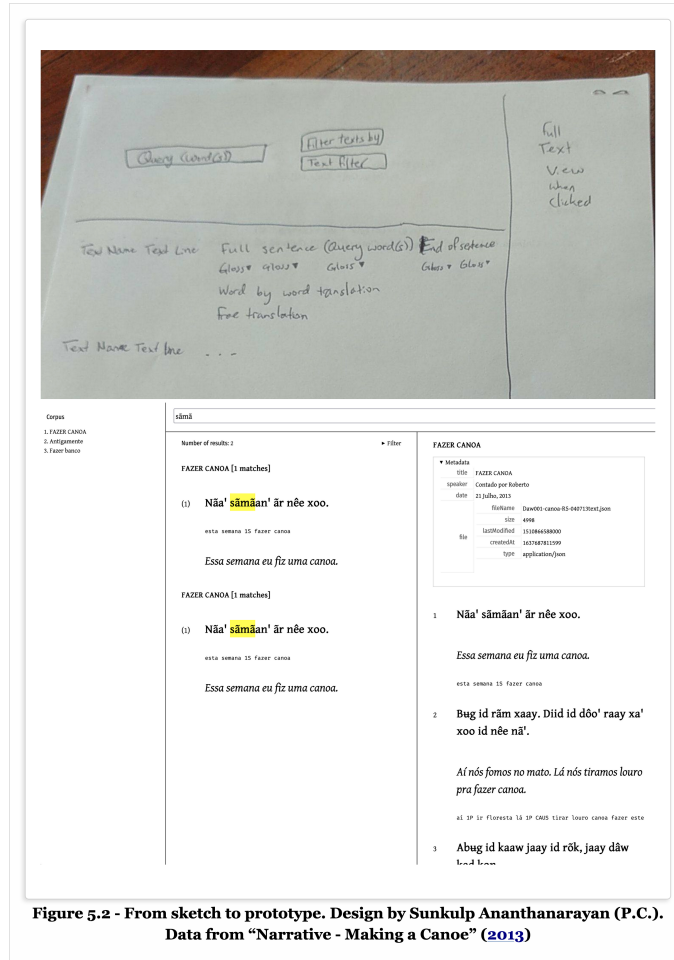
5.3.1 Designers

Linguists who design user interfaces to meet their documentation workflow needs. Linguists who choose to take on the role of *designers*. The middle layer — *designers* — is highlighted, as it corresponds to the target audience for the present work. Understanding the designer role is a prerequisite for (optionally) assuming an implementer role, but there are no prerequisites for users,

beyond familiarity with the use of particular applications. A typical participant at this level will be a working documentary linguist who has familiarity with current software and experience using it to create digital documentation of various kinds. These potential participants are those with a degree of fieldwork experience, and who may have experienced frustration in some aspects of their digital documentation work by incompatibilities in existing software tools. It is unrealistic to expect that all such linguists would choose to learn to program entire applications from the ground up — despite the fact that many have considerable technical expertise and even programming experience (often in languages which are adjacent to the approach described here such as R, Python, or Praat scripting). However, these linguists are in a good position to understand the approach demonstrated with the `docling.js` library. The data model will be easily understandable to them. What remains is to practice the skills of (1) imagining user interfaces that meet their needs, and (2) describing those imagined interfaces in such a way that they can actually be implemented. On some ideas for helping would-be interface designers, see the next section.

The approach of “sketching” designs is a well-developed aspect of web design (see Garrett ([2002](#)), Krug ([2013](#)), Buxton ([2007](#)), and Greenberg et al. ([2011](#))). The pair of images in the figure below include a simple design sketch (by Sunkulp Ananthanarayan) and an implemented prototype created by Ananthanarayan and this author. The prototype corresponds quite closely to the original design, which was intended to help make a disparate collection of interlinearized texts searchable. The prototype needs more iterative improvement before it could serve as a tool which is ready to be shared with the larger user community of general linguists, but

it met Ananthanarayan’s research goal, which was to search across a corpus of texts in different formats.



This is a good example of participating in interface design: 1) Ananthanarayan specified the desired workflow (querying across multiple texts); 2) the dataflows involved (filtering all the sentences from a corpus that matched a given query); and 3) the basic user interface desired to enable (1) and (2). These are learnable skills, but because our current “cultural” relationship to software in documentation is more oriented toward troubleshooting existing software than toward designing new

interfaces, such skills are not cultivated enough.

5.4.1 Implementers

Programmers who implement designs. This work has treated *web components* (custom HTML elements) as a kind of conceptual “black box”: we have considered design for documentation tools purely in terms of deciding how modular “components” function and to some extent how they be composed into more complex user interfaces. Of course, programming is unavoidably complex. The Web Platform offers certain advantages, however, to those who wish to understand better how client-side applications work. Most importantly, the web browser environment makes it possible to inspect how a user interface is constructed. In the screenshot below, a programmer has used a special command (Command-click on Mac, Ctrl+Shift+click on Windows/Linux) to *inspect* an individual button within an instance of the `<ipa-input>` component described in Chapter 4. In the lower part of the screenshot, the *developer tools* have opened, and the HTML markup of the page is highlighted to show the individual `<button>` tag which contains the `η` character.

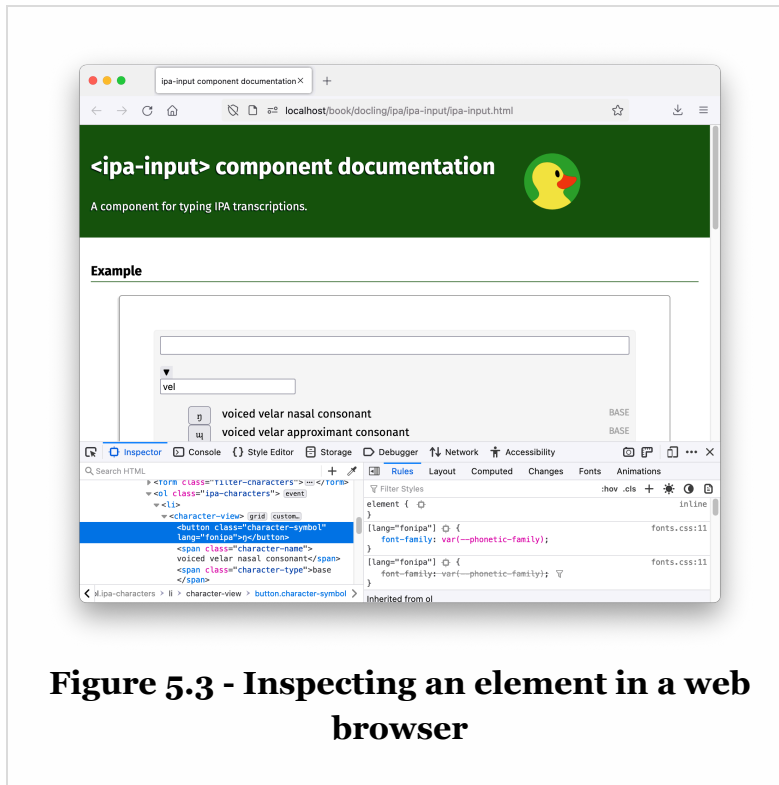


Figure 5.3 - Inspecting an element in a web browser

Developer tools are readily available in all modern web browsers, and they have many capabilities beyond simply matching rendered layout to HTML source. Javascript code can be run directly against pages for debugging purposes, CSS can be modified in real time, network requests and responses can be tracked, the accessibility of an interface can be evaluated, and so forth. `docling.js` was designed with a convention whereby every one of its web components exposes its data in such a way that it can be accessed directly from within the developer tools. This can aid developers in understanding the relationships between user interfaces and the “raw” data they are displaying and manipulating.

Using the developer tools is a primary means of coming to grips with how the various elements of the Web Platform function together. The same learning trajectory can be applied to learning how `docling.js` applications work, with a minimum of up-front configuration: the only necessary step is to open a web page containing a `docling.js` application, either locally or hosted on a server.

5.5.1 Users

The linguistics community in general. Even participating in the design of the user interfaces, let alone implementing user interfaces with code, requires a degree of time commitment. The demands on many working linguists may not allow them to take on the work of participating in an iterative process of user interface design. Nonetheless, it is hoped that if a community design process can be brought to a critical level of participation, then a steady stream of usable interfaces can be shared for use by the linguistics community at large. How exactly this would happen remains to be seen. Primarily it would involve participation by many more people than the current author. The scope of the present work, by focusing on the design layer, and demonstrating a functional working model of the implementation layer, aims to encourage linguists to realize that there is a clear path to becoming a productive participant in the design process, and that there is a range of degrees of participation possible. One might say that we're advocating a "choose your level of participation" model in the design process.

The goal of this 3-layer approach is to increase the number of participants at all levels. The `docling.js` software library is intended to be as transparent and self-

contained as possible — that is to say, no software libraries beyond the Javascript programming language itself is required in order to understand the code base in its entirety.^{[24](#)}

The functionalities of the Web Platform on which `docling.js` is built are standardized: that is to say, they are part of the same standards system that defines HTML, CSS, and Javascript itself. This minimizes risk moving forward, because whatever new standards emerge, web standards are defined as being “backwards compatible” — put simply, this means that whatever changes inevitably come to the definition of the web, the standards used to define `docling.js` will continue to be supported by all web browser manufacturers.

5.6.1 Documentation histories: Certainty and revision

Bowern (Bowern ([2007](#))) offered several criteria for evaluating manuscript fieldnotes, which offer an interesting point of comparison for “born-digital” user interfaces:

- When you cross something out, can you still read the rest of the word?
- When you write over a letter/word, can you tell which is the right representation and which is the wrong one?
- When you have two alternative transcriptions, can you tell which is right, or if they are both legitimate variants?
- Can you tell what is a deduction on your part and what was a metacomment made by your consultant?

- Can you tell what you need to check and what you know is right?
- Can you easily work out when these notes were written, who the consultant is, and what the notes are about?

Many of these problems are solved if a featural phonetic alphabet like the IPA can be used as the input orthography, but some are actually somewhat more difficult in the digital context. One of the features of manuscript fieldnotes is that the fieldworker has a vast array of annotative techniques that trace changes, corrections, reevaluations, and so forth, but if a correction is made in a digital document, corrections are lost. It is interesting to think through just what kinds of corrections are found on fileslips; the example below (Oswalt (1975)) is marked up with red rectangles to distinguish the various “content types” in the fileslip. Note that the upper left corner contains a note that says *check*. What is the digital equivalent of such an annotation? It might be profitable to explore adding a “status” tag to a word’s metadata (or even a grapheme or phoneme’s metadata) to track “processual” problems such as these.

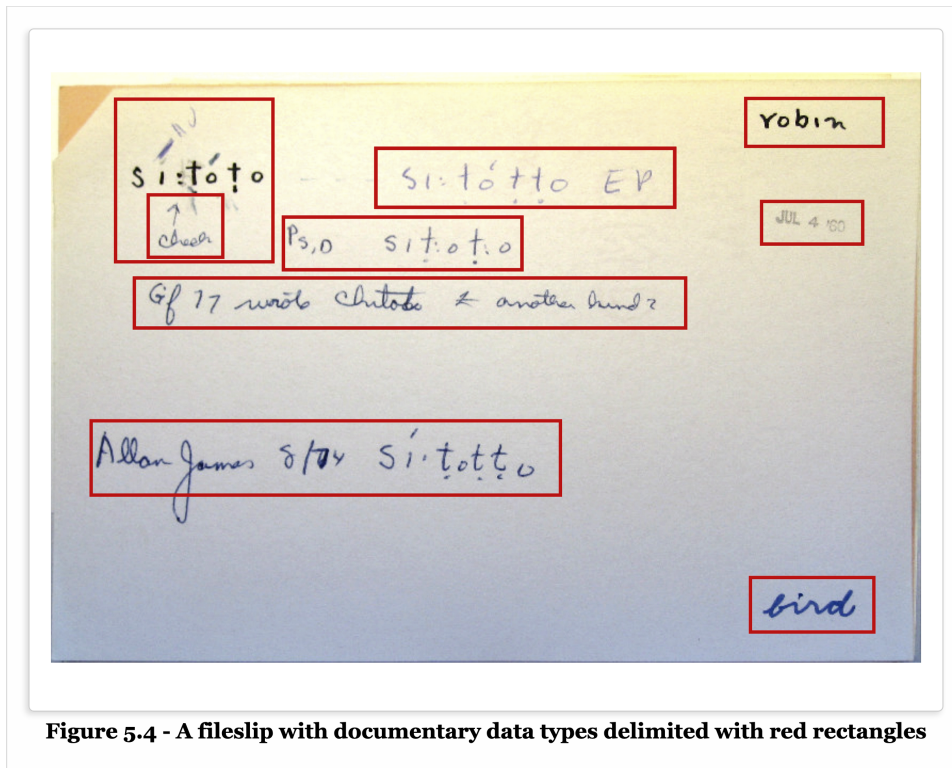


Figure 5.4 - A fileslip with documentary data types delimited with red rectangles

5.7 An open-access online course and next steps

In the coming year I will be creating a course based on the materials in this work: “Designing User Interfaces for Language Documentation with `docling.js`”. This course will delve deeper into the basics of implementation than the current work. The tentative outline is as follows:






Unit	Topic	Description
1	The Web Platform	What is it, and what does it offer for language documentation?
2	JSON	Modeling documentary data
3	HTML	Structured markup for structured documention
4	Templating	Rendering documentation in standard notations
5	Javascript	The Web Platform's built-in programming language
6	Web components	packaging functionality in custom HTML elements
7	Corpora	Interacting with Interlinear Texts
8	Lexicons	Managing words in the corpus
9	Grammar	Representing grammatical categories
10	Application design	From user interface components to full applications
11	Deploying	Examples of hosting docling.js applications on the web

Table 5.1 Course Outline

The course has several goals. One is to create an open resource online for any linguists who are interested in exploring the use of the Web Platform for language documentation. Another is to give the system a test-run: the hope is to develop documentation projects with several language communities which will serve as examples of organically designed projects that meet the needs of their creators.

As a primary outcome of this course, I plan to complete a set of user guides for all

of the components in the `docling.js` library. Thus far, some of the important guides have been written (see examples below), the documentation for the remaining components will be developed during the course, in collaboration with course participants.

-  [word-view guide](#)
-  [sentence-view guide](#)
-  [text-view guide](#)
-  [metadata-view guide](#)
-  [lexicon-view guide](#)

Interested readers might also wonder what steps they might take if they would like to get involved in the evolution of `docling.js` and the larger discussion of participatory design of software for language documentation.

- **Programmers** The source code of
- **Users**
- **Designers** You might consider

5.8 Conclusion

In this work I have described a simple, coherent, extensible, hierarchical data model which is sufficient for capturing a broad range of documentary data. I have described the design and implementation of prototype library of user interfaces in the form of web components, which are composable in the sense that they can be built up from

simple user interface into more complex “compound” applications. Such interfaces and applications can be designed to meet the needs of particular documentary workflows. It is the opinion of this author that our field is suffering from a kind of creeping institutionalization of what can be called “troubleshooting culture” in our relationship to technology: we put ever more effort into creating workarounds for software tools to force applications which were not designed to work together to meet all our needs, as opposed to designing new, custom tools specifically for our needs.

Language documentation is now a thoroughly digital discipline — there is no going back. We need to be intentional about how we approach both the form and construction of digital documentation moving forward. We are privileged to have the opportunity to work with many language communities who see value in language documentation. We must strive to ensure that our work is digitized in the most useful and maintainable way possible, so that our work may continue to protect the heart of the whole endeavor: human voices. It is incumbent upon us to not only help preserve their words, but also to ensure that we strive to create novel tools which can prioritize *their* needs, as well as those of academia.

Sapir said that language is “the most massive and inclusive art we know, a mountainous and anonymous work of unconscious generations.” It is interesting to consider which of the appellations about language also apply to language documentation. Unlike language itself, language documentation is neither unconscious nor anonymous. But like language itself, it is massive in scale, and sometimes overwhelming in its scope. To succeed, it must be inclusive, valuing the

contributions of all participants. A unified software system can play an important role in enabling those contributions. There is plenty of room for innovation, and perhaps there is even an opportunity to generate feelings of potency... and even joy? So, let us begin.

References

- Allsopp, John. 2000. "A Dao of Web Design." *A List Apart*. <https://alistapart.com/article/dao/>.
- Ameka, F. K., A. C. Dench, and N. Evans. 2006. *Catching Language: The Standing Challenge of Grammar Writing*. Vol. 167. Mouton de Gruyter.
- Berez, Andrea L., and Nicholas Thieberger. 2011. "Linguistic Data Management." *The Oxford Handbook of Linguistic Fieldwork*, 90–118.
- Bharatan, Vilma. 2009. "Custard Apple - Annona Squamosa." <https://www.flickr.com/photos/16454146@N06/3291420066/>.
- Bird, Steven. 2021. "Sparse Transcription." *Computational Linguistics* 46 (4): 713–44. https://doi.org/10.1162/coli_a_00387.
- Bird, Steven, and Gary Simons. 2003. "Seven Dimensions of Portability for Language Documentation and Description." *Language* 79 (3): 557–82. <http://emeld.org/workshop/2003/bowbadenbird-paper.pdf>.
- Boas, Franz. 1917. "Introductory." *International Journal of American Linguistics* 1 (1): 1–8. <http://www.jstor.org/stable/1263397>.
- Bow, Catherine, Baden Hughes, and S. G. Bird. 2003a. "A Four-Level Model for Interlinear Text."
- Bow, Catherine, Baden Hughes, and Steven Bird. 2003b. "Towards a General Model of Interlinear Text." In *Proceedings of EMELD Workshop*, 11–13. <https://www.linguistlist.org/emeld/workshop/2003/bowbadenbird-paper.pdf>.
- Bowern, Claire. 2007. "Online Notes to Accompany Linguistic Fieldwork: A Practical Guide." <https://web.archive.org/web/20081006040542/http://www.ruf.rice.edu/~bowern/fieldwork/Fieldnotes.pdf>.
- . 2015. *Linguistic Fieldwork: A Practical Guide*. Springer.

- Brugman, Hennie, and Albert Russel. 2004. "Annotating Multi-Media/Multi-Modal Resources with ELAN." In *LREC*. <https://pdfs.semanticscholar.org/3dac/d66abb9a06950017e057a14c911b592b1809.pdf>.
- Buxton, Bill. 2007. *Sketching User Experiences: Getting the Design Right and the Right Design*. 1st edition. Amsterdam Boston: Morgan Kaufmann.
- Chelliah, Shobhana L, and Willem J De Reuse. 2010. *Handbook of Descriptive Linguistic Fieldwork*. Springer.
- Comrie, Bernard, Martin Haspelmath, and Balthasar Bickel. 2015. "The Leipzig Glossing Rules: Conventions for Interlinear Morpheme-by-Morpheme Glosses." *Department of Linguistics of the Max Planck Institute for Evolutionary Anthropology & the Department of Linguistics of the University of Leipzig*. 28. <https://www.eva.mpg.de/lingua/pdf/Glossing-Rules.pdf>.
- Crowley, Terry. 2007. *Field Linguistics: A Beginner's Guide*. Edited by Nick Thieberger. Oxford: Oxford Univ. Press.
- Davis, Mark, and Andy Heninger, eds. 2020. "UTS #18: Unicode Regular Expressions." <https://unicode.org/reports/tr18/>.
- "Display - CSS: Cascading Style Sheets." 2021. Technical {Documentation}. *MDN Web Docs*. <https://developer.mozilla.org/en-US/docs/Web/CSS/display>.
- Dockum, Rikker, and Claire Bower. 2019. "Swadesh Lists Are Not Long Enough: Drawing Phonological Generalizations from Limited Data." *Lang. Document. Descript* 16: 35–54.
- Firth, John R. 1957. "A Synopsis of Linguistic Theory, 1930-1955." *Studies in Linguistic Analysis*.
- Forshaw, William. 2016. "Little Kids, Big Verbs: The Acquisition of Murrinhpatha

- Bipartite Stem Verbs.” {PhD} {Thesis}.
- Gair, James W., and W. S. Karunatilake. 1998. *A New Course in Reading Pāli*. Motilal Banarsidass Publ.
- Garrett, Jesse James. 2002. *The Elements of User Experience: User-Centered Design for the Web*. Indianapolis, Ind: Peachpit Pr.
- Gasser, Emily, and Claire Bower. 2014. “Revisiting Phonotactic Generalizations in Australian Languages.” *Proceedings of the Annual Meetings on Phonology* 1 (1). <https://doi.org/10.3765/amp.vii.17>.
- Gerdts, Donna B. 1998. “Incorporation.” *The Handbook of Morphology*, 84–100.
- Goodman, Michael Wayne, Joshua Crowgey, Fei Xia, and Emily M. Bender. 2015. “Xigt: Extensible Interlinear Glossed Text for Natural Language Processing.” *Language Resources and Evaluation* 49 (2): 455–85. <https://doi.org/10.1007/s10579-014-9276-1>.
- Greenberg, Saul, Sheelagh Carpendale, Nicolai Marquardt, and Bill Buxton. 2011. *Sketching User Experiences: The Workbook*. 1st edition. Amsterdam ; Boston: Morgan Kaufmann.
- H, Abdullah. 2016. “Sweetsop.” <https://www.flickr.com/photos/potent2020/29250675404/>.
- Hale, Kenneth. 2001. “Ulwa (Southern Sumu): The Beginnings of a Language Research Project.” *Linguistic Fieldwork*, 76–101.
- Hall, Patrick. 2012. “Hiligaynon Fieldnotes: Numerals.” Unpublished.
- Haspelmath, Martin. 2014. “The Generic Style Rules for Linguistics.” *Zenodo*. Doi 10.
- Heath, Jeffrey. 1982a. *Nunggubuyu Dictionary*. Australian Institute of Aboriginal

Studies.

———. 1982b. *Nunggubuyu Myths and Ethnographic Texts*. Australian Institute of Aboriginal Studies.

———. 1984. *Functional Grammar of Nunggubuyu*. Australian Institute of Aboriginal Studies.

Hegaret, Philippe le. 2011. “100 Specifications for the Open Web Platform and Counting W3c Blog.” <https://www.w3.org/blog/2011/01/100-specifications-for-the-open/>.

Henke, Ryan, and Andrea L. Berez-Kroeker. 2016. “A Brief History of Archiving in Language Documentation, with an Annotated Bibliography.” *Language Documentation & Conservation* 10 (December): 411–57.

“Hiligaynon Swadesh List.” 1966. Los Angeles, California.

http://archive.phonetics.ucla.edu/Language/HIL/hil_word-list_1965_01.wav.

Hill, Jane H. 2005. *A Grammar of Cupeño*. Vol. 136. University of California Publications in Linguistics. Univ of California Press.

Himmelmann, Nikolaus P. 2012. “Linguistic Data Types and the Interface Between Language Documentation and Description.” *Language Documentation* 6: 21.

Hopi Dictionary Project. 1998. *Hopi Dictionary/Hopiikwa Lavàytutuveni: A Hopi-English Dictionary of the Third Mesa Dialect*. University of Arizona Press.

Hsu, R. 1985. *Lexware Manual: Computer Programs for Lexicography Developed at the University of Hawaii*. University of Hawaii Department of Linguistics.

<http://www.montler.net/lexware/LexwareManual-RobertHsu.pdf>.

“HTML Living Standard.” 2021. Web standard. *HTML Standard*.

<https://html.spec.whatwg.org/multipage/>.

- IJAL. 2021. "International Journal of American Linguistics Style for the Formatting of Interlinearized Linguistic Examples." *International Journal of American Linguistics*. <https://www.americanlinguistics.org/wp-content/uploads/IJAL-interlinear.pdf>.
- Initiative (WAI), W3C Web Accessibility. n.d. "Web Content Accessibility Guidelines (WCAG) Overview." *Web Accessibility Initiative (WAI)*. Accessed September 28, 2020. <https://www.w3.org/WAI/standards-guidelines/wcag/>.
- "IPA i-Chart." 2021. https://www.internationalphoneticassociation.org/IPAcharts/inter_chart_2018/IPA_2018.html.
- Ishida, Richard. 2021. "IPA Character App 27." <https://r12a.github.io/pickers/ipa/>.
- Jacobson, Michel, Boyd Michailovsky, and John B. Lowe. 2001. "Linguistic Documents Synchronizing Sound and Text." *Speech Communication, Speech Annotation and Corpus Tools*, 33 (1): 79–96. [https://doi.org/10.1016/S0167-6393\(00\)00070-4](https://doi.org/10.1016/S0167-6393(00)00070-4).
- Katigbak, Sunny. 2009. "CALAMANSI." <https://www.flickr.com/photos/40223927@N08/4450918508/>.
- Klar, Kathryn. 2002. "John P. Harrington's Field Work Methods: In His Own Words." *Report of the Survey of California and Other Indian Languages, Proceedings of the 50th Anniversary Conference*, 12: 9–17. <https://escholarship.org/uc/item/7s82h46g>.
- Krug, Steve. 2013. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. 3rd edition. Berkeley, Calif.: New Riders.
- Ladefoged, Peter. 2003. *Phonetic Data Analysis: An Introduction to Fieldwork and Instrumental Techniques*. Wiley-Blackwell.

- Lambdin, Thomas O. 2006. *An Introduction to the Gothic Language*. Wipf; Stock Publishers.
- LSA. 2021. "Language Style Sheet." *Language: Journal Of The Linguistic Society Of America*. <http://www.linguisticsociety.org/files/style-sheet.pdf>.
- Manning, C. D., P. Raghavan, and H. Schutze. 2008. *Introduction to Information Retrieval*. Vol. 1. Cambridge University Press Cambridge.
- Marcotte, Ethan. 2010. "Responsive Web Design." *A List Apart*.
<https://alistapart.com/article/responsive-web-design/>.
- Meakins, Felicity, Jennifer Green, and Myfany Turpin. 2018. *Understanding Linguistic Fieldwork*. Routledge.
- Michaelis, Susanne Maria, Philippe Maurer, Martin Haspelmath, and Magnus Huber, eds. 2013. *Atlas of Pidgin and Creole Language Structures Online*. Leipzig: Max Planck Institute for Evolutionary Anthropology. <https://apics-online.info/>.
- Michailovsky, Boyd, Martine Mazaudon, Alexis Michaud, Séverine Guillaume, Alexandre François, and Evangelia Adamou. 2014. "Documenting and Researching Endangered Languages: The Pangloss Collection." *Language Documentation & Conservation* 8: 119. <https://halshs.archives-ouvertes.fr/halshs-01003734>.
- Milton [Gabaṅja], and Jeffrey Heath. 2018. "Olive Python and the Two Boys." In, edited by Simon Musgrave. Nunggubuyu Texts Online.
<http://users.monash.edu.au/~smusgrav/Nunggubuyu/texts/NMET-1.html>.
- Milton [Gabaṅja], Jeffrey Heath, and Simon Musgrave. 2018. "Ngambi 'Bathe'." *Nunggubuyu Online Dictionary*. <http://users.monash.edu.au/~smusgrav/>

Nunggubuyu/dictionary/ngambi.html.

Moran, Steven, and Michael Cysouw. 2018. *The Unicode Cookbook for Linguists*. Language Science Press. <https://doi.org/10.5281/zenodo.1296780>.

Moran, Steven, and Daniel McCloy, eds. 2019. *PHOIBLE 2.0*. Jena: Max Planck Institute for the Science of Human History. <https://phoible.org/>.

Moreland, Floyd L., and Rita M. Fleischer. 1990. *Latin: An Intensive Course*. Univ of California Press.

Munro, Pamela. 2001. "Field Linguistics." *The Handbook of Linguistics*, 130–49.

Musgrave, Simon, and Nick Thieberger. 2012. "Language Description and Hypertext: Nunggubuyu as a Case Study."

Nordhoff, Sebastian, ed. 2012. *Electronic Grammaticography*. LD&C Special Publication 4. University of Hawai'i Press. <http://hdl.handle.net/10125/4528>.

Ochs, E. 1979. "Transcription as Theory." *Developmental Pragmatics*, 43–72.

Oppenheim, A. Leo. 1968. "In Memoriam: Benno Landsberger 1890-1968." *Orientalia* 37 (3): 367–70. <https://www.jstor.org/stable/pdf/43074242.pdf>.

Oswalt, Robert L. 1975. "Kashaya Dictionary." <http://cla.berkeley.edu/item/2412>.

Palmer, A., and K. Erk. 2007. "IGT-XML: An XML Format for Interlinearized Glossed Texts." In *Proceedings of the Linguistic Annotation Workshop*, 176–83.

Raskin, Jef. 1994. "Intuitive Equals Familiar." *Communications of the ACM* 37 (9): 17–19.

Rice, Keren. 2011. "Documentary Linguistics and Community Relations." *Language Documentation & Conservation* 5: 187–207.

Roach, P. J. 1989. "Report on the 1989 Kiel Convention." *Journal of the International Phonetic Association* 19 (2): 67–80.

- Ruter, Weston. 2021. "International Phonetic Alphabet (IPA) Chart Unicode 'Keyboard'." <https://westonruter.github.io/ipa-chart/keyboard/>.
- S, Ingrid. 2016. "Starfruit." <https://www.flickr.com/photos/33682034@N00/29024081433/>.
- Sakel, Jeanette, and Daniel L. Everett. 2012. *Linguistic Fieldwork: A Student Guide*. Cambridge University Press.
- Samarin, W. J. 1967. *Field Linguistics: A Guide to Linguistic Field Work*. Holt, Rinehart; Winston.
- Sánchez, Roberto (speaker). 2013. "Narrative - Making a Canoe," Dâw Language Collection/La Colección del Idioma Dâw, July. <https://www.ailla.utexas.org/islandora/object/ailla%3A264379>.
- Sapir, Edward, Philip Sapir, and William Bright. 1900. *The Collected Works of Edward Sapir*. Berlin ; New York : M. de Gruyter. <http://archive.org/details/collectedworksof10sapi>.
- Seidel, Frank. 2016. "Documentary Linguistics: A Language Philology of the 21st Century." *Language Documentation and Description* 13: 23–63.
- Solnit, David B. 1997. *Eastern Kayah Li: Grammar, Texts, Glossary*. University of Hawaii Press.
- su-lin. 2007. "Inside the Rambutan." <https://www.flickr.com/photos/su-lin/1251225290/>.
- Suzuki, Shunryu. 2010. *Zen Mind, Beginner's Mind: Informal Talks on Zen Meditation and Practice*. Shambhala Publications.
- Szynalski, Tomasz. 2021. "Type IPA Phonetic Symbols - Online Keyboard." <https://ipa.typeit.org/full/>.

- Thieberger, N. 2004. "Documentation in Practice: Developing a Linked Media Corpus of South Efate." *Language Documentation and Description* 2: 169–78.
<https://minerva-access.unimelb.edu.au/handle/11343/34484>.
- (WHATWG), Web Hypertext Application Technology Working Group. 2020. "HTML Standard: Custom Elements." *HTML Living Standard*.
<https://html.spec.whatwg.org/multipage/custom-elements.html>.
- Wolfenden, Elmer. 1971. *Hiligaynon Reference Grammar*. Pali Language Texts: Philippines. PALI language texts: Philippines.
- Woodbury, Anthony C. 2011. "Language Documentation." In *The Cambridge Handbook of Endangered Languages*, edited by Peter K. Austin and Julia Sallabank, 159–86.
- . 2014. "Archives and Audiences: Toward Making Endangered Language Documentations People Can Read, Use, Understand, and Admire." *Language Documentation and Description* 12: 19–36.
- Yamada, Racquel-María. 2007. "Collaborative Linguistic Fieldwork: Practical Application of the Empowerment Model." *Language Documentation & Conservation* 1 (2).
- Zaefferer, D. 2006. "Realizing Humboldt's Dream: Cross-Linguistic Grammatography as Data-Base Creation." *Catching Language: The Standing Challenge of Grammar Writing*, 113.
-

1. Its use has become so widespread that in some cases it is even institutionalized: IJAL now requires that submissions to the *IJAL Texts Online* series be created with ELAN. <https://www.americanlinguistics.org/wp-content/uploads/preparing-ELANfiles.pdf>

2. This is a very general outline of the annotation process in ELAN; this description presumes a considerable amount of preliminary configuration: tiers must be configured, “linguistic types” must be specified, and so forth. Furthermore, at the very granular level of the user interface, much more could be said about the precise steps take in terms of clicking, dragging, keystrokes, and other actions required to achieve the task. All user interfaces requires conventions of this sort.[↵](#)
3. Again, the designers of ELAN are not to be faulted for such claims: the tool does what they claim it does.[↵](#)
4. Much more will be said about the mechanics of transcription in Chapter 4: transliteration, input methods, and so forth.[↵](#)
5. The interested reader who would like to gain a more in-depth understanding of HTML and related technologies may wish to refer to the “how-to” in [/howtos/html-basics/](#).[↵](#)
6. The possibility of using OCR (Optical Character Recognition) to expedite the process of converting print materials to digitized materials is feasible in some instances. In most cases, OCR must be post-edited, and if we are to extract structured information from the flow of text, further manual interventions may be necessary. The quality of the text recognition depends to a large degree on the quality of the input — for example, Heath’s typescript manuscripts have considerable noise, and would probably introduce enough errors that OCR output would have to be carefully post-edited.[↵](#)

7. The entire syntax of JSON may be expressed through the visual representation called a *railroad diagram*, as shown at <https://json.org>. A corresponding representation for the Javascript programming language would be many orders of magnitude more complex.↵
8. We should also note that the “whitespace” in these JSON texts is insignificant where it appears *outside* of quotes. Thus, as far as the computer is concerned, the following rather single-line, unindented recasting of the object view of the Belvedere data is completely identical to the first serialization as far as the computer is concerned: `{"year": 1958, "make": "Plymouth", "model": "Belvedere"}`. Indentation is helpful for human readers, however, and as data formats go JSON is fairly readable.↵
9. It is worth noting that Sapir himself considered that work incomplete: it was in “no sense a complete dictionary of the language but necessarily includes only such material as I happened to record.” ([Sapir, Sapir, and Bright 1900, 558](#))↵
10. Again, note that this data structure is a “skeleton” and contains only properties, not values — hence, the sample time stamps are empty. Examples will be seen in Chapter 3.↵
11. The terms *tag* and *element* are sometimes used interchangeably, but strictly speaking, the *tag* refers to the syntax as used in HTML as illustrated in [Figure 1.7](#) (e.g., `<tag>content</tag>`), whereas the *element* is the object which the browser creates in memory upon parsing an HTML page.↵

12. The term *attribute* unfortunately has many meanings in programming, and it is not used consistently. In the context of JSON data, we are using the term *attribute* to indicate a *property/value pair*. So for instance in the JSON object `{"form": "bentana", "gloss": "window"}`, the `form` and `bentana` constitute an attribute, with the string `form` functioning as a property, and `bentana` functioning as a value. The situation is analogous for `gloss` and `window`. In the context of HTML, we speak of *HTML attributes* as property/value pairs which are encoded inside of an HTML tag, and which modify the behavior or associated information of a tag in some way. The syntax for HTML attributes and property/value pairs in JSON objects is of course different: in an HTML attribute, the property (also sometimes referred to as the “name”) and its corresponding value are separated by an equals sign: `<entry-view src="bentana.json">`.↵
13. Note that in practice we do not need to import every custom element in the `docling.js` library individually — rather, we can simply import the `docling.js` file itself, which will make all the defined custom elements available at once.↵
14. The example for Eastern Kayah Li, it should be noted, is drawn from the English-Kayah index to the full Kayah-English glossary; in this case, simple one-word glosses and forms are distinguished by no other device than simple word delimitation — in the full glossary more complex formatting is employed.↵

15. Here, we will not delve into the details of exactly how the computer may be instructed to interpret the content of the JSON data, we simply accept as a premise the idea that once data is encoded in this way, it is possible to retrieve “parts” of the data programmatically, in a granular fashion. That is to say, once this data has been read by the computer, we may “ask” questions like “what is the transcription?” or “what is the form of the second word?”, and the computer will “know” to respond with the correct strings ("Hello , ako si Juan Lee ." and "ako", respectively.) The process of interpreting a textual data format such as JSON, HTML, or CSS is called “parsing” by computer scientists, in a much more limited way than it is generally used in linguistics.↵

16. Users of ELAN sometimes have to grapple with a similar problem, where the width of the text input is controlled by the width of the waveform selection, and word wrapping does not take place:



↵

17. Let us pause to appreciate the “dental bridge”. Ahem.↵

18. See Chapter 4 of Moran and Cysouw ([2018](#)).↵

19. For recent re-evaluations of appropriate sizes of basic vocabulary lists, see

Gasser and Bower (2014) and Dockum and Bower (2019).↵

20. I have included the number field under a metadata object as an aside — it is relevant because we are dealing with a numbered list, and that numbering might be important to preserve, for instance, for comparing with another list. We will ignore this metadata in the rest of the current discussion, but it is worth noting in passing that it is useful to think of the concept of “metadata” in a flexible way: it should be adaptable at any level of analysis as necessary (here, we are adding metadata at the word level).↵
21. Note that the grammatical category labels used here express just one analysis of the grammar verbal affixes in Hiligaynon and related Philippine languages. Such analysis is a topic of long-standing debate: which grammatical categories they are thought to express and the labels used to signify the various values of those categories is highly variable from source to source. For convenience, the analysis used here follows Wolfenden (1971)↵
22. Or more recently, its encrypted derivative, Secure HyperText Transfer Protocol, identifiable in URLs beginning with `https://`.↵
23. More commonly, individual Javascript files are “imported” from their own files, which reside alongside the HTML. This approach has the benefit that the same Javascript program can be imported into different HTML pages, and thus reused. Importing of this kind is also available with CSS stylesheets.↵
24. Thus, it is hoped that linguists with programming experience will be able to

follow the design of the code base and how to extend it without committing undue learning time toward learning additional Javascript-based software libraries or “frameworks” designed to enable the implementation of larger scale projects (currently popular Javascript frameworks at the time of writing include [ReactJS](#), [svelte](#), and [ember.js](#)).↵