The Text Alignment Network: Official Guidelines

The Text Alignment Network: Official Guidelines

Table of Contents

List of Tables

List of Examples

Part I. General Overview

Table of Contents

Chapter 1. Introduction

Definition and audience

The Text Alignment Network (TAN) is a suite of XML encoding formats and associated rules and recommended practices, intended to serve anyone who wishes to encode, exchange, and study varieties of text reuse: translations, paraphrases, adaptations, summaries, quotations, and so forth.

The XML encoding formats behind TAN have been designed to be maximally readable and editable by both humans and machines, to be useful independent of any programs or tools, and to be both syntactically and semantically interoperable. TAN files are suitable not only for importing and exporting but, in many cases, for direct use. Built upon stand-off annotation, the TAN format allows scholars to edit and study the same texts independently and collaboratively.

TAN is meant primarily to support research in linguistics, translation studies, and the humanities—any scholarly field concerned with the interpretation of textual reuse or versions of texts, whether written as logosyllabaries (e.g., Han Chinese), abugidas (e.g., Tagalog), abjads (e.g., Syriac), or alphabets (e.g., English), and whether composed in antiquity, the recent past, or any time between (periods covered by, e.g., classics, Biblical studies, medieval studies, early modern studies, or twenty-first-century studies). Although expressive of scholarly nuance and complexity, TAN files are meant to benefit everyone, to be used in broad applications such as multilingual publishing, language learning, and machine translation.

Rationale and purpose

Translations, quotations, paraphrases, and so forth—all forms of text reuse—are important sources for scholars. Some texts have been lost in their original form and can be studied only through later translations, paraphrases, or quotations. And even when an original survives, these later versions are often worth study, revealing as they do something of the genius or idiosyncrasies of those who translated or quoted the original, which in turn sheds light on how words, concepts, and works were preserved or altered, shaped or combined across the generations and cultures who read and circulated the versions.

The comparative study of versions of texts requires, as a foundation, some way to align words, sentences, paragraphs, and other text segments. Such alignment can be challenging. Identifying which words or phrases in a translation correspond to which words or phrases in the original might result in complex, overlapping sets. And even larger segments such as sentences and paragraphs may not line up well. Further, every version of a text is part of a much larger, complex history of text reuse, and proper study of that context requires not only multiple versions of multiple works, but collaboration across projects and fields of study.

The Text Alignment Network aims to provide an XML encoding format that permits the exchange of detailed independent scholarship on versions of texts. TAN files adopt a syntax suitable for humans to read and edit, expressive enough to allow scholars to register doubt and nuance, and sufficiently structured to permit complex computer-based queries across independent datasets. The format is designed to allow a person to concentrate on a set of tasks associated with a specific activity (editing, grammatical analysis, word alignment, etc.) without having to worry about other types of analysis. The format compels editors to declare their views or assumptions about language and texts in a structured manner, so that other users of the data (both human and computer) can determine whether the data is suitable for their needs. Because nearly all TAN data must be expressed in both human- and computer-readable forms, the information can be used in semantic web applications.

TAN has been designed to support specific research questions such as the following:

- How do I encode a text I wish to share in such a way that it is likely to align with other versions of that text?
- How do I encode quotations of one work in another such that the data might apply to multiple versions of either work?
- How do I share texts, alignments, or other linguistic data such that anyone using the data will be aware when I make corrections or changes to the material?
- How do I align multiple versions of a single work when those versions may not match very well, or when the reason for alignment may be vague or ambiguous?
- How do I make detailed alignments between a source and its translation, when there may be messy many-to-many relationships, and where I might need to express doubt or alternative possibilities of alignment?
- How do I create a detailed alignment between a source and its translation such that, once completed, the dataset can be interrogated to find out how a certain class of words (roots, part of speech, etc.) gets translated in the target language, and with what frequency?
- How do I answer the above questions such that my data can be used interoperably by others?

The last question is especially significant, because it is expected that as persons or projects share or publish their TAN-compliant texts, there will emerge a decentralized corpus of texts along with stylesheets, tools, and applications that use TAN texts. As this TAN-compliant corpus grows, its interoperability will allow the repertoire of research questions to expand beyond any single-project corpora, to allow scholars to fruitfully investigate broader, comparative questions such as:

- For classical Greek texts, how were words with the root -ιστημι ("stand") translated into ancient Latin? In what specific ways did the vocabulary of technical terms shift from pre-Christian translations into later, Christian ones?
- How do the reformed Chinese translation technique of Sanskrit Buddhist texts, attested by Dao An (312-385 CE), compare to reforms in the seventh and eighth centuries of Syriac translations of Greek texts?
- How do Arabic translations of Greek texts from the Abbasid period differ from those of Sanskrit?
- Can an anonymous English translation of a modern French novel be identified with known translators of French novels from the same period?
- How do present-day translations of official United Nations documents differ across languages?

These ambitions behind these questions should be tempered by cautions about what TAN is not intended to do:

- TAN is not a tool. It is a format. Just as other file formats, such as JPEG, CSV, RTF, and so forth, TAN does not dictate what tools or applications should be used to create, edit, or display TAN-compliant files.
- TAN does not create alignments or, on its own, answer research questions. It merely lays the foundation for such questions to be investigated. Separate applications, stylesheets, and other tools must be developed to use the format for research.

- TAN is not meant to support linguistic data beyond its scope (defined and explained in these guidelines). That is, TAN is a supplement, not a replacement, for other markup formats such as TEI, docbook, and so forth, or other alignment formats such as XLIFF or TMX.
- TAN is not meant to be the most computationally efficient format. XML by its very nature sacrifices terseness—a cardinal virtue for computational efficiency—in favor of human readability. TAN is meant, first and foremost, to support lossless, interoperable exchange in a format transparent to human editors and readers.

Design Principles

To facilitate the research questions mentioned above, the TAN encoding formats and this manual have been designed around a few core principles.

Scholarly freedom: Scholars should be able to create data within their sphere of inquiry simply, expressively, independently, and with fidelity to their guiding lights.

- Given two ways of expressing the same idea, simplicity is better than complexity, expressiveness than silence. Simplicity and expressiveness should be treated as complementary ideals. In cases where one must be chosen over the other, simplicity is to be preferred.
- Editors should not have to supply data that is redundant, irrelevant to the immediate points of inquiry, or more reliably and authoritatively found elsewhere.
- Editors should be able to register doubt about claims. If in doubt about an assertion, an editor should be able to state alternatives.
- Editors should be able to work on the same material indepedently but interoperably.
- Editors should work freely within their theories, opinions, and assumptions about language. They should declare those positions, not suppress or alter them.

Scholarly responsibility: Scholars must make their data uniquely citable, and responsibly describe how that data was created.

- Each TAN file should have an expressive, unique, persistent name that can be cited and used independent of the file's location or availability.
- Editors must supply, at the very minimum, the core statements of responsibility that are expected in scholarly communication generally, i.e.:
  - What was done by whom, when.
  - What sources have been used.
  - Who holds rights over the data, and what reuse is permitted.
  - What editorial assumptions and decisions were made in creating the data.

Human-friendly digital utility: Computers should be able to import, process, and create human-friendly data reliably, consistently, and interoperably.

- Each human-readable datum should have a computer-readable counterpart, to make the material suitable for linked data (semantic web).
- Given multiple methods of referring to textual units or linguistic concepts, the most human readable method is to be preferred.

- All classes and types of formats in the TAN suite should be structured consistently and predictably.
- Validation and recommended best practices should depend upon stable technologies or standards.
- Each TAN file should be integrally complete, independent of any organization or third-party software such as text processors or version control software.

Chapter 2. Starting off with the TAN Format

If you are new to markup languages, or if you are unfamiliar with technical terms and acronyms such as *XML*, *RDF*, *XPath*, and so forth, you should start with this chapter, which uses a simple example to illustrate the steps typically taken to create and edit TAN files. By the end of this chapter, you be sufficiently oriented to create and edit a simple corpus of TAN transcriptions and alignments.

The discussion touches on a number of concepts, some of which may be new. These concepts will be introduced, but only briefly. If you need more explanation, you should follow some of the suggestions for further reading.

Creating TAN Transcription and Alignment Data

Let us take a simple example, that of aligning two English versions of the nursery rhyme *Ring-a-ring-a-roses*, sometimes known as *Ring around the Rosie*. Our goal here is to publish two versions of the nursery rhyme in the TAN format so that they are most likely alignable with any other TAN version of the poem that might be published.

We begin by finding the versions we want. In this case we use exemplars published in 1881 [http://lccn.loc.gov/12032709] and 1987 [http://lccn.loc.gov/87042504], the former from the UK the latter from the US. Each of these books have other rhymes, but we've already decided to focus upon one particular nursery rhyme, so we transcribe them and nothing else:

Table 2.1. Ring around the Rosie

1881 (UK) version	1987 (US) version
Ring-a-ring-a-roses, A pocket full of posies; Hush! Hush! Hush! Hush! We're all tumbled down.	Ring-a-round the rosie, A pocket full of posies, Ashes! Ashes! We all fall down.

We must be sure to save each of the two transcriptions as plain Unicode text, preferably with `.xml` at the end of each file name. If you use a word processor (Word, OpenOffice, Google Docs, and so forth) you must be certain to save the file as plain text, in Unicode encoding (not Western European or other encodings). But even then you have to beware, because these programs can be too sophisticated for our work and sometimes generate erroneous formats. Word processors are in fact a bit too powerful for our purposes. We will be working with raw text, and will not be altering colors, fonts, margins, and so forth. Much better for our work is a text editor [http://en.wikipedia.org/wiki/Text_editor]. And even better still would be an XML editor [http://en.wikipedia.org/wiki/XML_editor], which does the same thing a text editor does, but does other things too, such as tell us when our XML file is invalid, or provide shortcuts that save much typing.

Note

Software suitable for your needs comes in many styles and prices. In addition to the links provided above, you may wish to visit the comparative lists for both text editors [http://en.wikipedia.org/wiki/Comparison_of_text_editors] and XML editors [http://en.wikipedia.org/wiki/Comparison_of_XML_editors]. Some XML editors are so powerful they can be very confusing to use at first. Be certain to take advantage of tutorials and documentation associated with any XML editor you begin to use.

Our first task is to get these two versions into separate files with the appropriate markup. Each TAN transcription file has two major parts: a head and a body. For now, we focus on only the second part, the body, as well as a few the necessary preliminary lines that stand above both the head and the body. First, the 1881 (UK) version:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-T.rnc" type="appl
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-T.sch" type="appl
<TAN-T xmlns="tag:textalign.net,2015:ns" id="tag:parkj@textalign.net,2015:ring01">
    <head>
    . . . . . . .
    </head>
    <body xml:lang="eng" in-progress="false">
        <div type="line" n="1">Ring-a-ring-a-roses,</div>
        <div type="line" n="2">A pocket full of posies;</div>
        <div type="line" n="3">Hush! Hush! Hush! Hush!</div>
        <div type="line" n="4">We're all tumbled down.</div>
    </body>
</TAN-T>
```

And now the 1987 (US) version:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-T.rnc" type="appl
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-T.sch" type="appl
<TAN-T xmlns="tag:textalign.net,2015:ns" id="tag:parkj@textalign.net,2015:ring02">
   <head>
   . . . . . . .
   </head>
   <body xml:lang="eng" in-progress="false">
      <div type="l" n="1">Ring-a-round the rosie,</div>
      <div type="l" n="2">A pocket full of posies,</div>
      <div type="l" n="3">Ashes! Ashes!</div>
      <div type="l" n="4">We all fall down.</div>
   </body>
</TAN-T>
```

These are standard eXtensible Markup Language (XML) files. (If you are already familiar with this format you may wish to skip ahead.) XML is rather simple. It provides a way to take a text or a collection of data and give it some structure through markup. In the examples above, the markup is in boldface.

Each file begins with a prolog, marked by the lines that begin with `<?`. The first line in each prolog simply states that what follows is an XML document. The next two lines point to the files that will be used to check to see whether or not our data is valid. For now we will skip the specific details. These three lines will be identical, or nearly so, from one TAN transcription file to the next, and we can simply cut and paste when we want to start a new one.

The fourth line is the opening tag of what is called the root element, here called `<TAN-T>`. That opening tag, `<TAN-T...>` is answered by a closing tag, `</TAN-T>`, the last line. The same thing happens with all the other elements in this example. `<head>` is answered by `</head>`, `<body>` by `</body>` and each `<div...>` by `</div>`. These elements nest within or alongside each other, but they never overlap. (The prohibition on overlapping elements is one of the cardinal rules of XML.) This relationship means that every XML file can be thought of as a tree, with the root at the trunk and the enveloped elements as branches, terminating in metaphorical leaves. It is helpful to use the tree metaphor when we describe the path we take, either toward the leaves or the root. In this manual, we may use the terms *rootward*, *rootmost*, *leafward*, and *leafmost* when we want to trace movement within an XML document.

An XML document is also profitably thought of as a family tree, a metaphor that provides terminology that will be very common. In our examples above, `<TAN-T>` is the *parent* of `<body>`, and `<body>` the parent of the four `<div>` elements. Vice versa, each `<div>` is the *child* of `<body>`, and `<body>` is the child of `<TAN-T>`. Distant parental relationships can be described with the terms *ancestor* and *descendant*. `<TAN-T>` is the ancestor of every element it encompasses, and every element encompassed by `<TAN-T>` is its descendant. Paratactic relationships are also important. `<head>` and `<body>` are *siblings* to each other, and every `<div>` is a sibling to every other `<div>`.

Inside of the opening tags for the `<TAN-T>`, `<body>`, and `<div>` elements are pairs of text joined by an equals sign. The left side of each pair is called an attribute, and on the right side, within the quotation marks, is the value. `<TAN-T>` has two attributes, `@xmlns` and `@id` (attributes are commonly referred to with @ before the name). We will skip `@xmlns` for now; this attribute specifies the namespace of the XML file, a somewhat advanced topic.

The value of `@id`, however, is quite important and our first item of business. Every TAN file has an `@id` that uniquely and permanently identifies the file itself. It is quite similar to the name we give a file when we save it, and to the names we see when we browse through our files, except that it should not be changed. When we make corrections to our work, we will not change the `@id` value. We simply note a change (to be discussed later) with the date of the change.

The value of `@id` is always what is called a tag uniform resource name (tag URN). It always starts with `tag:`, followed by an email address or domain name that we owned on a given day. (It is okay to use an obsolete address.) After that email address or domain name comes a comma (no spaces) and a date on which we owned it, in the international standard format of year, month, and date, joined by hyphens, e.g., 2014-12-31. If we leave off a day value, it is assumed to be the first of the month; if we leave off the month value it is assumed to be January. In the examples above, `[USER@DOMAIN.NET],2014` indicates that the email address was owned on the stroke of midnight (Coordinated Universal Time) January 1, 2014. After that comes a colon, and then any name we wish to assign to the file.

Note

Many if not most TAN files will be circulated under a liberal license, encouraging people to modify and use the data freely, with attribution. If you decide to modify someone else's TAN file, and you become responsible for changes, not the original person or organization. Your first point of order should be to change the email address or domain name at the heart of the tag URN, substituting one that you own or owned. The element `<succeeds>` in the `<head>` allows you to credit the original. If you do not make this change, you are incorrectly passing the work off as someone else's.

We have anticipated a simple collection of texts, so we've called the files `ring01` and `ring02`. (If we run out of names, or want to restart, we can simply use a new email-date preface, e.g., `[USER@DOMAIN.NET],2014-01-02`.)

The element `<body>` contains our transcription. `@xml:lang`, required, specifies the principal language of the transcribed text. We use the standard 3-letter abbreviation for English. (See later in the guide for more complex language requirements.) By saying that `@in-progress` is `<false>`, we indicate that we have finished our transcription and have no further plans to develop it. It doesn't mean that the file is error free. We will have the option to make corrections later. It just means that we have done all the work that we intended, and any further changes will be restricted to corrections of errors. This attribute is optional; if left off, the TAN file is assumed to be a work in progress.

Our transcription has been divided into four `<div>` elements. How we divide up the work is entirely up to us. But we must make sure that every bit of text is enclode by a leafmost `<div>`. That is, every `<div>` must be the parent of only other `<div>`s, or none at all. We cannot have a `<div>` that mixes any text or other elements with other `<div>`s. The values of `@type` and `@n` indicate, respectively, the type of division and the name of the division. We have used `line` in the first example, but we could easily have also used `l` (as we did in the second) or `ln` or any other phrase that we think will make intuitive sense to other users (we will see why below). We have used arabic numerals for the values of `@n`, but the value, once again, could have been anything. But since other people may use our work, we have adopted the naming scheme that seems most common and expected.

Aside from the `<head>` element (discussed later), that's all we need in the transcription. We can now move to alignment.

There are two different types of alignment, one emphasizing breadth, the other, depth. The broad type of alignment, called TAN-A-div, allows us to specify TAN transcriptions of as many versions of as many works as we wish, and to fine-tune the alignment upon the basis of the `<div>` elements within the transcription. We do not specify why we wish to align the versions. We only declare our interest in doing so. The type of alignment emphasizing depth, called TAN-A-tok, allows us to take any two (and no more) TAN transcriptions and specify word-to-word (or better put, token-to-token) relationships, and specify what type of relationship holds between them. TAN-A-div is suitable for work that focuses on the general alignment of multiple versions of one or more works at a single time. TAN-A-tok is for highly detailed, precise alignment of two text versions.

For our example, we start with a TAN-A-div file (once again suppressing `<head>`):

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-A-div.rnc" type="
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-A-div.sch" type="
<TAN-A-div xmlns="tag:textalign.net,2015:ns" id="tag:parkj@textalign.net,2015:ring
    <head>
    . . . . . . .
    </head>
    <body/>
</TAN-A-div>
```

In the prolog, the first line is identical to the first line of our transcription files. The second and third lines are identical, aside from pointing to the validation files for alignment. Even the fourth line looks like the transcription file, other than the new name for the root element, `<TAN-A-div>`, and the new value for `@id`.

The penultimate line, `<body/>`, is what is called an empty element, and is equivalent to `<body></body>`. Collapsing the opening and the closing tags of the element into a single tag provides a shorthand syntax for elements contains nothing. It will become apparent, when we discuss `<head>` below, why our `<body>` can be empty.

The other kind of alignment, TAN-A-tok, takes a bit more work, because we must first identify words that correspond with each other. Even before we do that, we need to decide what kind of relationship holds between the two texts. Let us pretend, for the sake of illustration, that the 1987 version is a direct descendant (and therefore variation) of the 1881 version. We therefore see our task as creating a map of how the older version became the newer one. There are different ways of approaching this task and deciding on the specifics is not as straightforward as some might think. We will also assume for this example that punctuation is both irrelevant and a marker between words, better termed in this context as *tokens* (*word* is notoriously difficult to define, and the substitute *token* lacks many undesirable implications).

We now create a TAN-A-tok file:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-A-tok.rnc" type="
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-A-tok.sch" type="
<TAN-A-tok xmlns="tag:textalign.net,2015:ns" id="tag:parkj@textalign.net,2015:TAN-
    <head>
    . . . . . . .
    </head>
    <body bitext-relation="B-descends-from-A" reuse-type="adaptation" in-progress=
        <!-- Examples of picking tokens by number -->
        <align>
            <tok src="ring1881" ref="line 1" ord="1"/>
            <tok src="ring1987" ref="l 1" ord="1"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 1" ord="2"/>
            <tok src="ring1987" ref="l 1" ord="2"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 1" ord="3"/>
            <tok src="ring1987" ref="l 1" ord="3"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 1" ord="4"/>
            <tok src="ring1987" ref="l 1" ord="4"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 1" ord="5"/>
            <tok src="ring1987" ref="l 1" ord="5"/>
        </align>
        <!-- Examples of picking tokens by value -->
        <align>
            <tok src="ring1881" ref="line 2" val="A"/>
            <tok src="ring1987" ref="l 2" val="A"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 2" val="pocket"/>
            <tok src="ring1987" ref="l 2" val="pocket"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 2" val="full"/>
            <tok src="ring1987" ref="l 2" val="full"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 2" val="of"/>
            <tok src="ring1987" ref="l 2" val="of"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 2" val="posies"/>
            <tok src="ring1987" ref="l 2" val="posies"/>
        </align>
        <!-- Examples of picking tokens by number with commas, hyphens, and 'last'
        <align>
            <tok src="ring1881" ref="line 3" ord="1, 2"/>
            <tok src="ring1987" ref="l 3" ord="1"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 3" ord="3 - 4"/>
            <tok src="ring1987" ref="l 3" ord="2"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 4" ord="1"/>
            <tok src="ring1987" ref="l 4" ord="1"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 4" ord="2"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 4" ord="3"/>
            <tok src="ring1987" ref="l 4" ord="2"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 4" ord="last-1"/>
            <tok src="ring1987" ref="l 4" ord="last-1"/>
        </align>
        <align>
            <tok src="ring1881" ref="line 4" ord="last"/>
            <tok src="ring1987" ref="l 4" ord="last"/>
        </align>
    </body>
</TAN-A-tok>
```

Once again, the first four lines, the prolog and root element, should look familiar, with the only significant changes being the names of the validation files, the name of the root element (`<TAN-A-tok>`) and the value of `@id`.

The heart of the data is `<body>`, which has, in addition to `@in-progress`, two more attributes, `@reuse-type`, which specifies the default type of relationship between the two sources, and `@bitext-relation`, which specifies how the versions relate to each other. Our two values, `B-descends-from-A` and `adaptation`, are arbitrary names that we define in the `<head>` (discussed later).

`<body>` is the parent of one or more `<align>` elements, each of which correlates a set of tokens in the two texts through its `<tok>` children. Each `<tok>` has, in this example, three attributes. `@src` takes a nickname (an `@id` reference) that points to one of the two transcriptions; we have used `ring1881` and `ring1987` but we could have just as easily used anything else such as `uk` and `us`. `@ref` has a value that points to a specific `<div>` in the source transcription; and `@ord` or `@val` specify which token is intended, either by word number (`@ord`) or text of the actual word (`@val`). Either technique is fine, and can be mixed, as in the example. You may also notice that the comma and hyphen can be used in `@ord` to point to multiple words within the same `<div>`, and that `last` and `last-X` (where `X` is a digit) can be used to point to a word token relative to the last one in a `<div>`.

Each `<align>` can establish one-to-one, one-to-many, many-to-one, or many-to-many relationships between the two texts. It is perfectly fine to have the same word represented in multiple `<align>` elements. And if an `<align>` has `<tok>` elements belonging to only one source, such as in the fourth-to-last `<align>` above, we have what is called, in these guidelines, a *half-null alignment*. That is to say, using our example, that we have stated that the second word of line four of the 1881 version is excluded from the act that we have called `adaptation` (which is, as we shall see, defined in the `<head>`). If this were a translation, it would be as if we were saying that this word was excluded from the translation. (If this was a half-null alignment for the later source, we would say that it was a word that the translator added.) A half-null alignment is not to be confused with our own silence. If we nowhere mention a token in our TAN-A-tok file, and if we specify through `@in-progress` that our work is finished, then we are saying only that we have nothing to say about it.

We could have aligned the two texts in different ways. Perhaps further study will reveal that we were in error to associate the second "ring" with "round" is line 1. We can make corrections, even after publication, and signal the change to users of our data. There are also ways to express doubt or alterative opinions. We can even correlate fragments of tokens (letters, prefixes, infixes, or suffixes). All these more advanced uses are discussed in later chapters of these guidelines.

For now, we have finished everything for two TAN transcriptions and two alignment files, except for the `<head>`. Before getting into details, we need first to discuss (1) the principles behind the TAN `<head>` and (2) how to ensure that the data in `<head>` is both human readable and computer readable.

The Principles of TAN Metadata (<head>)

Unlike `<body>`, which carries the raw data, `<head>` contains what is oftentimes called metadata. That is, `<head>` takes data that describes the data. Because the TAN format is intended primarily to serve scholars, some metadata requirements are stricter than those in other formats. Scholars who use our data really need to know some essential things before thoughtfully studying and handling the data we produce. For example, what are the sources we have used? Who produced the data? When? What key assumptions have been made in producing the data? What rights do other people have to use the data? The questions are not difficult to answer, but they are critical.

Some of these questions are specific to certain types of data, and others can be applied universally, no matter what kind of data. We have seen that the TAN transcription `<body>`s look quite different from those in the two alignment files. And there are other kinds of TAN files that cover other types of data, such as lexicomorphology and tokenization rules. And TAN may grow to cover other kinds of information. So `<head>` has been concisely and predictably structured so it can apply across all TAN formats. This reduces potential confusion and makes it easier for other people using our data to find the information they want.

The rigorous scholarly requirements for TAN metadata are offset somewhat by another principle that was adopted in the design of TAN, namely, that each format's `<head>` should focus upon describing the data in `<body>` and not other things. That is to say, for the two transcriptions in our example above, we should definitely indicate what our sources are. But we should not do what librarians do so much better and try to write a catalog entry describing the books we have used. Anyone who really wants that information should go elsewhere. We need merely point to those other places. The TAN format does not forbid us from providing this information. But the `<head>` is designed so that we can stay focused on the task and data at hand.

TAN was also designed with the assumption that all metadata should be useful to both humans and computers. For our example above, we must describe the work we have chosen in such a way that *Ring around the Rosie* is comprehensible not just to the reader but to the computer, using syntax that a computer can act upon.

Take for example the 1881 book we have used for our first transcription. For the human reader we can say simply something like "Kate Greenaway, *Mother Goose*, New York, G. Routledge and sons [1881]". But computers need a more controlled, predictable syntax before they can be directed to the correct edition of *Mother Goose*. The human-readable string is too complex, and syntactically opaque. Perhaps a better identifier for a computer would be international standard book numbers (ISBNs), which not only distinguish the 1984 version illustrated by Kayoko Okumura Cussler's novel from the 1984 version illustrated by William Joyce but can be used to distinguish the various types of editions: hardback, paper, electronic, and so forth. The ISBNs for the Okumura version, 0671493159, and for Joyce's, 0394865340, can be converted into a machine-actionable string called universal resource names (URNs), in this case `urn:isbn:0-671493159` and `urn:isbn:0-394865340`. (Our 1881 version was published before the ISBN program was introduced, but we will see below other ways to name it.)

URNs are formalized naming schemes that are regulated by a central body (Internet Assigned Numbers Authority, IANA) to ensure that people and organizations can legitimately coin and use permanent, persistent, unique names for various types of things. Ther are URN schemes for journals (by means of ISSNs), articles (DOIs), and movies (ISANs), thereby allowing one to refer to them with a unique string that a computer can process.

All URNs are simply names. They don't tell you where an object is, just what its name is. To provide a unique *location*, however, we have universal resource locators (URLs), which might be much more familiar from daily use of the Internet, e.g., `http://academia.edu`. Like URNs, URLs are also centrally regulated, with individuals or organizations buying the rights to domain names from a central registry (usually through a third-party vendor).

Both URNs and URLs can be thought of as the same type of thing, namely, a universal resource identifier (URI), sometimes called an international resource identifier (IRI) to make clear that any alphabet in Unicode, not just Latin, may be used. URIs/IRIs are, in essence, nothing more than the set of all URNs and URLs. These four acronyms can be easily confused, and it is best to disambiguate them by thinking of the last letter in each. URIs/IRIs Incorporate both Locators and Names.

IRIs are essential to a system frequently called the semantic web or linked (open) data, an agreed way of writing and processing data that relies upon IRIs and a simple data model to connect them. The semantic web allows independent parties to make assertions about things, and if they happen to use the same IRI vocabulary to describe those things, computers can make associations between disparate, heterogenous datasets. This allows computers to make connections that humans might not make on their own. For example, a computer could be used to plot the geographical distribution of specific books in different library systems, if it has access to published IRIs in different online library catalogs.

TAN has been designed to be linked-data friendly, and so requires in its head almost all data to be representable not just in a human readable form but also as an IRI, so computers can act upon the data. (It is this requirement that makes possible another principle mentioned earlier, that we should provide metadata about our data and not about other things.)

Our first task, then, in writing the `<head>` sections of our four TAN files is to look for IRI vocabulary that will be familiar to the community of practice most likely to use our files. In trying to find suitable IRIs, we will find that the persons, things, and concepts we want to describe will be range from generally familiar to generally unfamiliar.

Generally familiar: The two books that provide the basis of our transcription are well catalogued and generally known. A number of services provided by librarians provide a controlled IRI vocabulary that can be used by anyone to describe uniquely a particular version of a book. WorldCat [http://www.worldcat.org] (run by OCLC) and the Library of Congress [http://catalog.loc.gov] are good examples. In our case, we have found accurate Library of Congress IRIs for both editions of *Mother Goose*: `http://lccn.loc.gov/12032709` and `http://lccn.loc.gov/87042504`. Observe that these two IRIs are, more specifically, URLs. If we paste these strings into our browser, we retrieve a record that describes the book. But this locator has not taken us to the book per se, only to information *about* the book. That is because, under the principles of linked data, a person or entity who owns a domain name can mint URLs based on that domain name *also as names for objects*. This subtle point can sometimes be confusing to those who are new to the semantic web, because

URLs conflate locations and names. Much of the motivation for this decision is convenience: by giving an object a name that is also a URL, you can then set up a server to tell a computer or a human more information about the object that is given that URL's name.

Let's now find an IRI to name the work we have chosen, *Ring around the Rosie*. The work is widely known, and even has a Wikipedia entry [http://en.wikipedia.org/wiki/Ring_a_Ring_o%27_Roses]. That Wikipedia entry is fortuitous. The Universities of Leipzig and Mannheim and Openlink Software have collaborated on a project called DBPedia [http://wiki.dbpedia.org/About], which is committed to providing a unique URN for every Wikipedia entry in the major languages. The DBPedia URN for the work we have chosen is `http://dbpedia.org/resource/Ring_a_Ring_o%27_Roses`. Once again, this is both a name and a locator. It names a specific intangible object, namely a nursery rhyme that we've called *Ring around the Rosie*, no matter what specific version. But if you put that name into your browser, you will get back more information about that named object.

Familiar, but only in small circles: We will need to have names for some of the people who edited the file. Most people who contribute to the creation of the data file will not be well-known, public figures. If they are, and if they are famous enough to have a Wikipedia entry, then a DBPedia IRI could be used. Or if some of the contributors are also published authors, there is a good chance that they are listed in the databases of either VIAF [http://viaf.org] or ISNI [http://isni.org], both of which publish unique IRIs for persons. If we find in those systems an IRI for someone who created or helped edit the file, we can use it.

There is a good chance, however, that many contributors will not be listed in these general databases. In these cases, we can assign our own IRI to name these participants. We have already done something like this by assigning tag URNs to our four transcriptions (the value of `@id` in the rootmost element). We can do the same for our editors. If a student Robin Smith has been helping with proofreading, we can take an email address for Robin (even one that doesn't work any more) and a date when the email was sed and construct a tag URN such as `tag:smith.robin@example.com,2012:self`. This has a slight drawback in that we cannot type this string into our browser to find out more about the Robin, but it at least allows us to assign a name that will not be confused as the Robin Smith identified by ISNI as `http://isni.org/isni/0000000043306406`. If we wish to go a step further, we could mint a URN from a domain name that we own, and set up a linked data service that offers more information, human- and computer-readable, about Robin, but this is not required.

Another example of field-specific IRIs is the concept of relationship between two text-bearing objects. We are assuming for the sake of illustration that the version published in the 1987 *Mother Goose* is a direct descendant of the 1881 version. Our assumption is important to declare, because if we had a different view on how one related to the other, it would probably affect the specifics of our word-for-word alignments. Because no suitable IRI vocabulary yet exists for such concepts, TAN has coined an IRI that can be used by anyone wishing to declare that the second of two sources descends from the first through an unknown number of intermediaries: `tag:textalign.net,2015:bitext-relation:a/x+/b`.

We face a similar issue when thinking about text reuse. We generally consider the 1987 version to be an adaptation of the 1881 version. And there are not stable, well-published IRI vocabularies for text reuse. So we adopt a TAN-coined IRI, `tag:textalign.net,2015:reuse-type:adaptation:general`.

For other examples of IRIs coined by TAN, see iris.xml.

Generally unfamiliar: Some things or concepts will be unknown to very few people, perhaps only to us. If we plan to refer to that thing or concept often, it is preferable to coin a tag URN, as described above. But in some cases, we might find that a tag URN we minted for some concept or thing was, in hindsight, misleading or poorly constructed, because we hadn't taken into account other things that should be named. So if we wish to avoid these kinds of situations, we can assign a random IRI called a universally unique identifier (UUID), e.g., `urn:uuid:3fd9cece-b246-4556-b229-48f22a5ae2e0`. These uuid URNs, which are generated by computers through randomizing functions, are very useful. The likelihood that a randomly generated uuid will be identical to any other uuid is astronomically improbable, making them reliable, uniquely names for anything (barring someone copying and reusing that uuid URN to name some other object or concept). Numerous free UUID generators exist, and can be found through Intenet searches.

To humans, a UUID on its own is meaningless, and rather ugly. But it is a good start. We always have the option, later, of adding an IRI. It's perfectly fine to give one object or concept multiple IRIs. But the reverse is never true. One should never use the same IRI to identify more than one object or concept.

Creating TAN Metadata (<head>)

Now that we have explored various IRI vocabularies for concepts around our versions of *Ring-a-ring-a-roses*, we can now complete the metadata in our four TAN files. Let us start with the TAN-T file of the 1881 version:

```
<head>
    <name>TAN transcription of Ring a Ring o' Roses</name>
    <master-location>ring-o-roses.eng.1881.xml</master-location>
    <rights-excluding-sources rights-holder="park">
        <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
        <name>This data file is licensed under a Creative Commons Attribution
            License. The license is granted independent of any rights and lice
            associated with the source. </name>
    </rights-excluding-sources>
    <source>
        <IRI>http://lccn.loc.gov/12032709</IRI>
        <name>Kate Greenaway, Mother Goose, New York, G. Routledge and sons [1
    </source>
    <declarations>
        <work>
            <IRI>http://dbpedia.org/resource/Ring_a_Ring_o%27_Roses</IRI>
            <name>"Ring a Ring o' Roses" or "Ring Around the Rosie"</name>
        </work>
        <div-type xml:id="line">
            <IRI>http://dbpedia.org/resource/Line_(poetry)</IRI>
            <name>line of poetry</name>
        </div-type>
        <recommended-tokenization which="general-1"/>
    </declarations>
    <agent xml:id="park" roles="creator">
        <IRI>tag:parkj@textalign.net,2015:self</IRI>
        <name>Jenny Park</name>
    </agent>
    <role xml:id="creator">
        <IRI>http://schema.org/creator</IRI>
        <name xml:lang="eng">creator</name>
    </role>
    <change when="2014-08-13" who="park">Started file</change>
</head>
```

The `<head>` element is the parent of eight children, all but one required in every TAN-T file.

`<name>` is the human readable form of the `@id` that is inside the root element, `<TAN-T>`. It can be anything. And we can supply more than one `<name>`, in case we wish to provide it in different languages or variations.

`<master-location>` is mandatory only if we have claimed through `@in-progress` that the file is no longer in progress. One or more of these elements provide URLs where master versions of the file are kept (and updated). They may be absolute URLs, such as an address on the Internet, or it may be a relative URL, in case we are working exclusively on our local computer. During validation, if a file does not match the one in the master version, a warning is returned. This allows us to publish a file, make corrections, and notify other users without having to keep track of who may be using the file. Similarly, if we are working with a copy of a file and it doesn't match the version in the master location, a warning will be returned, along with a message or a location of the elements that were last changed.

`<rights-excluding-sources>` contains information about rights to the data we are releasing. This element has nothing to do with the copyright of the source we have used (although, having been published in 1881, the book is clearly in the public domain). This once again gets to the TAN metadata principle of describing our data and not other things. We have the option to describe the license of the source we have used (see the rest of the guidelines for guidance), but we absolutely must declare whether we have placed additional scrictures on the dataset we have created. In this example, we have released the data under a creative commons license. The child element `<IRI>` specifies the IRI assigned by Creative Commons, and `<desc>` describes it in human-readable format.

The conjunction of `<IRI>` and `<desc>`, the *IRI + name pattern*, is a recurrent feature of TAN files. We may include any number of `<IRI>` or `<desc>` elements in an IRI + name pattern. But if we do so, we are stating that they are all synonymous. The all name the same thing, not different things.

`<source>` points, through its IRI + name pattern, to a computer- and human-readable description of the book we have chosen.

`<declarations>` contains data that is specific to the type of TAN file in hand, to declare the assumptions we have made in creating the data. In this case, because we are working with transcriptions, we have three major components: `<work>`, `<div-type>`, and `<recommended-tokenization>`.

`<work>` uses the IRI + name pattern to name the work we have chosen to transcribe. `<div-type>` specifies the type of divisions we have chosen to use to segment the transcription. In a more complex text, there would be several `<div-type>`s. Each one has an `@xml:id`, which takes as a value some nickname that we wish to use for `@type` values of `<div>`s. `<recommended-tokenization>` specifies a pattern recommended for changing the text into word tokens. In the example above, the `<recommended-tokenization>` is empty, and specifies through `@which` that a general tokenization pattern should be used, which means that word tokens should be created by dividing according to space, and each point of punctuation should be treated as a word. Other general options for `@which` include `general-words-only-1` (all punctuation is suppressed) and `precise-1` (tokens should be formed only from spaces or U+200B, ZERO WIDTH SPACE). Tokenization also allows the IRI + pattern for customized TAN tokenization files (TAN-R-tok), discussed elsewhere in these guidelines.

The IRI + name pattern is also used for `<agent>`, which describes who was involved in creating the data, and `<role>`. Any number of `<agent>` and `<role>` elements may be included and combined as we wish. The `agent` in this case, Jenny Park, has been given a tag URI. The `<IRI>` value of `<role>` comes from the vocabulary of schema.org [http://schema.org], which is maintained by

Bing, Google, and Yahoo! in conjunction with the W3C (the nonprofit organization dedicated to universal Internet standards), but we could have used Dublin Core or some other IRI vocabulary describing behaviors, responsibilities, and roles.

Remember that `<head>` is focused on the data, not its sources, so the claim that Jenny Park is the creator pertains only to the data. No inference should be made about who created the source. If someone wants that information, or anything else about the source, they should pursue the identifier we have provided under `<source>`.

`<change>` has attributes `@when` and `@who` that specify who made the change/comment and when. The value of `@when` is always a date plus optional time formatted according to the standard `YYYY-MM-DD` + time (optional).

Note

If you are working with an XML editor that supports Schematron Quick Fixes then whenever you are at a `@when` or any other attribute that requires an ISO date and the value is malformed you will be given a prompt to insert today's date or date and time in a conformant manner. Using this feature will save you time and effort.

`@who` always carries a value that refers to an `agent/@xml:id`. Both `<change>` (as well as `<comment>`, missing here) lack any IRIs, mainly because the likelihood that the data would ever be reused, repeated, or linked to is altogether too remote to be make a mandated `<IRI>` useful.

The other transcription file looks similar:

```
<head>
  <name>TAN transcription of Ring around the Rosie</name>
  <master-location>ring-o-roses.eng.1987.xml</master-location>
  <rights-excluding-sources rights-holder="park">
     <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
     <name>Creative Commons Attribution 4.0 International License</name>
     <desc>This data file is licensed under a Creative Commons Attribution 4.0
        License. The license is granted independent of rights and licenses ass
        source. </desc>
  </rights-excluding-sources>
  <source>
     <IRI>http://lccn.loc.gov/87042504</IRI>
     <name>Mother Goose, from nursery to literature / by Gloria T. Delama, 198
  </source>
  <declarations>
     <work>
        <IRI>http://dbpedia.org/resource/Ring_a_Ring_o%27_Roses</IRI>
        <name>Ring around the Rosie</name>
     </work>
     <div-type xml:id="l">
        <IRI>http://dbpedia.org/resource/Line_(poetry)</IRI>
        <name>line</name>
     </div-type>
     <recommended-tokenization which="general-words-only-1"/>
     <filter>
        <normalization>
           <IRI>tag:textalign.net,2015:normalization:hyphens-discretionary-rem
           <name>discretionary hyphens suppressed</name>
```

```
         </normalization>
      </filter>
   </declarations>
   <agent xml:id="park" roles="creator">
      <IRI>tag:parkj@textalign.net,2015:self</IRI>
      <name xml:lang="eng">Jenny Park</name>
   </agent>
   <role xml:id="creator">
      <IRI>http://schema.org/creator</IRI>
      <name xml:lang="eng">creator</name>
   </role>
   <change when="2014-10-24" who="park">Started file</change>
   <comment when="2014-10-24" who="park">See p. 39 of source.</comment>
</head>
```

One significant difference is that `<declarations>` has a new child, `<filter>`, which contains a `<normalization>` statement that declares, in both a description and an IRI, that we have opted to remove word-break line-end hyphenation. This provides a cautionary note to users of our data who might value line-end hyphenation. Any number of `<normalization>`s can be used to describe any alterations we might have made in our transcription. In other transcriptions we could use this feature to declare other suppressions, such as editorial comments or footnote signals.

Note that the value of `div-type/@xml:id` here, the letter `l`, differs from our previous transcription file, `line`. Even though we have adopted a different nickname, they are treated as equivalent because in each file we have defined `l` or `line` with the same IRI, `http://dbpedia.org/resource/Line_(poetry)`. A computer that later aligns these two files will not worry so much about `l` and `line`, but will look at the underlying IRI that defines these terms. This exemplifies how linked data (see above) can support our work. We are free to use abbreviations that make sense to us, yet tie those abbreviations into the larger infrastructure by means of IRIs. It also means that our transcriptions can be assigned divisions that may be generally rare and unfamiliar or common but only to a specific field (e.g., sections of a legal document).

Our alignment `<head>`s will look slightly different. We start with the TAN-A-div alignment file:

```
<head>
   <name>div-based alignment of multiple versions of Ring o Roses</name>
   <master-location>ringoroses.div.1.xml</master-location>
   <rights-excluding-sources rights-holder="park">
      <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
      <name>Creative Commons Attribution 4.0 International License</name>
      <desc>This data file is licensed under a Creative Commons Attribution 4.
         License. The license is granted independent of rights and licenses as
         source. </desc>
   </rights-excluding-sources>
   <source xml:id="eng-uk">
      <IRI>tag:parkj@textalign.net,2015:ring01</IRI>
      <name>Transcription of ring around the roses in English (UK)</name>
      <location when-accessed="2015-03-10">../TAN-T/ring-o-roses.eng.1881.xml<
   </source>
   <source xml:id="eng-us">
      <IRI>tag:parkj@textalign.net,2015:ring02</IRI>
      <name>Transcription of ring around the roses in English (US)</name>
      <location when-accessed="2014-08-13">../TAN-T/ring-o-roses.eng.1987.xml<
   </source>
   <declarations/>
```

```
        <agent xml:id="park" roles="creator">
           <IRI>tag:parkj@textalign.net,2015:self</IRI>
           <name xml:lang="eng">Jenny Park</name>
        </agent>
        <role xml:id="creator">
           <IRI>http://schema.org/creator</IRI>
           <name xml:lang="eng">creator</name>
        </role>
        <change when="2014-08-14" who="park">Started file</change>
     </head>
```

Much of this will look similar to the previous two examples. Every alignment file has only one kind of source, namely TAN transcription files, nothing else. Therefore `<source>`'s `<IRI>` always takes the `@id` value in the TAN transcription file. `<name>` may replicate exactly the title found in the transcription file, or it may be modified, perhaps to harmonize better with the descriptions of the other texts aligned in the file. `<source>` also has an child element not seen in the earlier two examples, `<location>`, which specifies where the digital file was accessed and when (through `@when-accessed`). We may include as many of these `<location>` elements as we wish (starting with a preferred or reliable location). The `@when-accessed` value is important, because if the owner of the file later edits it, and provides a date for the change, the validator will be able to detect that our alignment file depends upon an older version and return a warning with details about the latest change. We can then explore the source and see if the alterations merit any changes on our part.

Our TAN-A-div file could have any number of `<source>`s, and not necessarily for the same work. It also does not matter in which order we put the `<source>`s. By putting the 1987 or the 1881 version first, we are not implying that one is the source of the other. The most important consequence is that most processes that use our alignment file will treat the first-listed source as the basis from which to compare other sources.

`<declarations>` is empty, mainly because we have, in this case, no working assumptions to declare. In more advanced uses, this element would not be empty.

This `<head>` explains why the `<body>` of our TAN-A-div file could be empty. In the `<head>` we have already specified the texts that would need to be aligned and where they are to be found. All TAN-A-div files assume, by default, that every source should be aligned with each other upon the basis of `<div>`s, especially leaf `<div>`s. That is, the TAN-A-div format assumes all alignments are to be made implicitly unless otherwise specified.

In fact, a TAN-A-div file is in this case technically unnecessary. The two files are already predictably structured and aligned, and the works and division types are defined identically. But we will see that the options available in a TAN-A-div's `<declarations>` and `<body>` will allow us not only to deal with inconsistencies in source transcriptions but to create interesting alignments that go well beyond the automatic alignment afforded by identically structured transcriptions.

Meanwhile we turn to our fourth file, the TAN-A-tok alignment, whose `<head>` looks like this:

```
     <head>
         <name>token-based alignment of two versions of Ring o Roses</name>
         <master-location>ringoroses.01+02.token.1.xml</master-location>
         <rights-excluding-sources rights-holder="park">
             <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
             <name>Creative Commons Attribution 4.0 International License</name>
             <desc>This data file is licensed under a Creative Commons Attribution
                License. The license is granted independent of rights and licenses
```

```
            source. </desc>
    </rights-excluding-sources>
     <source xml:id="ring1881">
        <IRI>tag:parkj@textalign.net,2015:ring01</IRI>
        <name>Ring o roses 1881</name>
        <location when-accessed="2015-01-17">../TAN-T/ring-o-roses.eng.1881.xm
    </source>
    <source xml:id="ring1987">
        <IRI>tag:parkj@textalign.net,2015:ring02</IRI>
        <name>Ring o roses 1987</name>
        <location when-accessed="2015-01-17">../TAN-T/ring-o-roses.eng.1987.xm
    </source>
    <declarations>
        <bitext-relation xml:id="B-descends-from-A">
            <IRI>tag:textalign.net,2015:bitext-relation:a/x+/b</IRI>
            <name>B descends directly from A, unknown number of intermediaries
            <desc>The 1987 versions is hypothesized to descend somehow from th
                1881 version, mainly for the sake of illustration.</desc>
        </bitext-relation>
        <reuse-type xml:id="adaptationGeneral">
            <IRI>tag:textalign.net,2015:reuse-type:adaptation:general</IRI>
            <name>general adaptation</name>
        </reuse-type>
        <tokenization src="ring1881 ring1987" which="general-1"/>
    </declarations>
    <agent xml:id="park" roles="creator">
        <IRI>tag:parkj@textalign.net,2015:self</IRI>
        <name xml:lang="eng">Jenny Park</name>
    </agent>
    <role xml:id="creator">
        <IRI>http://schema.org/creator</IRI>
        <name xml:lang="eng">creator</name>
    </role>
    <change when="2015-01-20" who="park">Started file</change>
</head>
```

The TAN-A-tok `<head>` looks similar to the previous examples, except that `<declarations>` has three children.

`<bitext-relation>` states through an IRI + name pattern the stemmatic relationship we think holds between the two sources. (Stemmatics is the study of the chain of transmission by which the versions written by authors made their way into the copies, versions, and editions that are extant; it frequently involves the creation of genealogical-like tree to depict how text-bearing objects with different copies of the same text derive from or relate to each other.)

One or more `<reuse-type>`s specify how one text has reused another. The IRI we have used shows that we suspect that the text reuse is general adaptation. If this were a translation or a quotation or some other kind of text reuse, we would have used a different IRI value.

A third declaration, `<tokenization>`, specifies the tokenization pattern we have adopted to define our word tokens. It is much like the `<recommended-tokenization>`s that feature in TAN-T files. Note, however, that `@src` may take more than one value, to specify that the same tokenization rule should be applied to both sources. We could have applied different tokenization patterns had we wanted to, either to customized TAN-R-tok files or to one of the `<recommended-tokenization>` patterns in the source.

Aligning across Projects

We expand our example now and imagine what it might be like to extend the alignment outside our project. Let us assume that we have found in the library of another project a TAN transcription of a work that looks quite similar to our own:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-T.rnc" type="appl
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-T.sch" type="appl
<TAN-T xmlns="tag:textalign.net,2015:ns" id="tag:hans@beispiel.com,2014:ringel">
   <head>
      <name>TAN Transkription, Ringelreihen mit Riederfallen</name>
      <master-location>http://beispiel.com/TAN-T/ringel.xml</master-location>
      <rights-excluding-sources rights-holder="schmidt">
         <IRI>http://creativecommons.org/licenses/by/4.0/</IRI>
         <name>Creative Commons Namensnennung 4.0 International Lizenz.</name>
         <desc>Dieses Werk ist lizenziert unter einer Creative Commons
            Namensnennung 4.0 International Lizenz.</desc>
      </rights-excluding-sources>
      <source>
         <IRI>http://www.worldcat.org/oclc/4574384</IRI>
         <name>Franz Magnus Böhme, Deutsches Kinderlied und Kinderspiel: Volksüber
            allen Landen deutscher Zunge, gesammelt, geordnet und mit Angabe der Q
            1897.</name>
      </source>
      <declarations>
         <work>
            <IRI>tag:beispiel.com,2014:texte:holderbusch</IRI>
            <name>"Die Kinder auf dem Holderbusch"</name>
         </work>
         <version>
            <IRI>urn:uuid:31648039-3dbb-49b9-b66e-9bd2cd11630e</IRI>
            <name>zweite Version</name>
         </version>
         <div-type xml:id="Zeile">
            <IRI>http://dbpedia.org/resource/Gedichtzeile</IRI>
            <name>Gedichtzeile</name>
         </div-type>
         <recommended-tokenization which="general-1"/>
         <filter>
            <normalization>
               <IRI>tag:kalvesmaki@gmail.com,2014:normalization:hyphens-discretion
               <name>Keine Bindestriche</name>
            </normalization>
         </filter>
      </declarations>
      <agent xml:id="schmidt" roles="Produzent">
         <IRI>tag:hans@beispiel.com,2014:selbst</IRI>
         <name xml:lang="eng">Hans Schmidt</name>
      </agent>
      <role xml:id="Produzent">
         <IRI>http://schema.org/producer</IRI>
         <name xml:lang="eng">Produzent</name>
      </role>
      <change when="2014-08-13" who="schmidt">Anfang</change>
      <comment when="2014-08-13" who="schmidt">unten auf der Z. 438, recht</commen
   </head>
   <body xml:lang="deu" in-progress="false">
      <div type="Zeile" n="a">Ringel, Ringel, Reihe!</div>
      <div type="Zeile" n="b">Sind der Kinder dreie,</div>
      <div type="Zeile" n="c">Sitzen auf dem Holderbuch,</div>
      <div type="Zeile" n="e">Schreien alle: husch, husch, husch!</div>
   </body>
</TAN-T>
```

It seems clear to us that this 19th-century German version is quite similar to our two English versions. We have some alignment options open to us. Two more sets of word-for-word alignments would be interesting, but remember, just because we find a text that nicely aligns with others does not mean that we *must* align them. In this case, we choose not to worry about word-for word alignments, and we focus here only on the TAN-A-div alignment, so that, for example, we can later generate an HTML report that will allow us to more conducively read the three versions in parallel and study their relationships.

To that end, we first observe some differences between this transcription and our other two. First, the value of `<work>` is not the one we have given our two versions. Second, the `<div-type>` is defined as `http://dbpedia.org/resource/Gedichtzeile`. Third, the lines have been lettered instead of numbered. And last, the editor seems to have made a typographical error, making the last line `n="e"` instead of `n="d"`). These four differences typify some of the inconsistencies that are found as people work independently on similar texts.

Note

There are a few other differences in this third transcription that do not affect our alignment. `<version>` is used to distinguish different version of the same work found on the same text-bearing object. Notice that the `<IRI>` value is a uuid, in this case because the editor was not prepared to deploy a formal IRI naming scheme (perhaps using a tag URN) that would be satisfactory for work-versions.

These are points we can easily reconcile in our TAN-A-div file, which we now expand to include the German version. We make the following adjustments (in boldface):

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-A-div.rnc" type="
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-A-div.sch" type="
<TAN-A-div xmlns="tag:textalign.net,2015:ns" id="tag:parkj@textalign.net,2015:ring
    <head>
       <name>div-based alignment of multiple versions of Ring o Roses</name>
       <master-location>ringoroses.div.1.xml</master-location>
       <rights-excluding-sources rights-holder="park">
          <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
          <name>Creative Commons Attribution 4.0 International License</name>
          <desc>This data file is licensed under a Creative Commons Attribution 4.
             License. The license is granted independent of rights and licenses as
             source. </desc>
       </rights-excluding-sources>
       <source xml:id="eng-uk">
          <IRI>tag:parkj@textalign.net,2015:ring01</IRI>
          <name>Transcription of ring around the roses in English (UK)</name>
```

```
         <location when-accessed="2015-03-10">../TAN-T/ring-o-roses.eng.1881.xml<
      </source>
      <source xml:id="eng-us">
         <IRI>tag:parkj@textalign.net,2015:ring02</IRI>
         <name>Transcription of ring around the roses in English (US)</name>
         <location when-accessed="2014-08-13">../TAN-T/ring-o-roses.eng.1987.xml<
      </source>
      <source xml:id="ger">
         <IRI>tag:beispiel.com,2014:ringel</IRI>
         <name>Transcription of an ancestor of Ring around the roses in German</n
         <location when-accessed="2014-08-22">http://beispiel.com/TAN-T/ringel.xm
         <location when-accessed="2014-08-22">../TAN-T/ring-o-roses.deu.1897.xml<
      </source>
      <declarations/>
      <agent xml:id="park" roles="creator">
         <IRI>tag:parkj@textalign.net,2015:self</IRI>
         <name xml:lang="eng">Jenny Park</name>
      </agent>
      <role xml:id="creator">
         <IRI>http://schema.org/creator</IRI>
         <name xml:lang="eng">creator</name>
      </role>
      <change when="2014-08-14" who="park">Started file</change>
      <change when="2014-08-22" who="park">Added German version.</change>
   </head>
   <body>
      <equate-works sources="eng-uk ger"/>
      <equate-div-types>
         <div-type-ref src="ger" div-type-ref="Zeile"/>
         <div-type-ref src="eng-uk" div-type-ref="line"/>
      </equate-div-types>
      <realign>
         <anchor-div-ref source="ger" ref="Zeile e"/>
         <div-ref source="eng-us" ref="l 4"/>
      </realign>
   </body>
</TAN-A-div>
```

The first major change is the insertion of a new `<source>`, identifying the name and location of the third example. Note that two locations have been provided, one for the original location and another for the copy saved locally into our project folder. Validation will occur at the first document available, so if we wanted to work primarily off our local copy, we would have put it first. By placing it second, we indicate that we want to have our local copy as a backup, in case the first is not available.

The second major insertion is a new `<change>`, documenting when we made the alterations. The value of `@when` effectively alters the version number of our TAN-A-div file.

The third major change populates the `<body>` with elements that calibrate the new version to the other two. `<equate-works>` says that, for the sake of this alignment, the works defined in the UK version and the German version to be considered equivalent. We did not mention the US version because we do not need to. As was mentioned above, anyone using a TAN-A-div should assume that all alignments are transitive unless otherwise specified. That includes specifications like this. If A and B are already defined to be the same work, and we specify that A and C are also the same work, then it must be inferred that B and C are as well. Note, we are not committing ourselves to the proposition that they are in reality the same work. We are making this statement only provisionally, to facilitate the alignment.

`<equate-div-types>` declares that what the German version calls Zeile is, for the sake of this alignment, equivalent to what the UK version calls line. Transitivity means that Zeile is inferred to be equivalent to what the US version calls `l`.

A `<realign>` takes care of the apparent typographical error, this time using the US version as the main way to correlate the division references. Any `<div-ref>` in a `<realign>` is wrested from automatic alignment. If there is an `<anchor-div-ref>` then it will be realigned with that reference and, by the law of transitivity, anything that aligns to it, in this case the UK version.

We do not need to reconcile the `@n` values of `a`, `b`, and `c` in the German version. The TAN format allows four numeration systems other than Arabic numerals: Roman numerals (uppercase or lowercase), alphabetic numerals (a, b, c, ..., z, aa, bb, ...), and digit-alphabet combinations (e.g., 1a, 1e, 4g) or alphabet-digit combinations (e.g., a4, a5, b5). All these will be treated as their numerical equivalent.

With these changes, the new version is completely synchronized with the other two. Our work may have been simplified if we had just modified the German version ourself. But then we would not have been able to connect our work to other TAN files that may be depending upon this version. Perhaps we should have gotten in touch with Hans Schmidt, the producer of the German version, and suggest that he modify the version to make it align better. In the case of `<div-type>`, he need merely add another line: `<IRI>http://dbpedia.org/resource/Line_(poetry)</IRI>`. This line, in addition to the preexisting `<IRI>`, specifies that the two IRIs are equivalent. Perhaps he has reasons for labeling the lines with letters, and perhaps he is reluctant to explicitly identify this poem with *Ring around the Rosie*. That is within his rights. (Remember, TAN is meant to provide a framework within which opinions can be registered, even against popular opinion.) But the conversation might lead to our pointing out that `n="e"` should probably be `n="d"` and that there is an apparent discrepancy in the last line (the original, printed book has the poem twice on page 438, one with "Holderbuch," the other, "Holderbusch"). In correcting the original transcription and providing a record of the change through `<change>`, he tacitly notifies anyone else using the file that corrections have been made.

There is much more to TAN, of course. The rest of these guidelines provide full documentation. In addition, other examples in the TAN repository provide ideas on how to develop your other alignments, perhaps more complex. The data files in that repository are open for modification, copying, reuse, and so forth. Just remember, if you do that, be certain to change the tag URN of each TAN file's `@id` so that it points to an email address or domain name that you own(ed).

Chapter 3. Intermediate Use of the TAN Format

This chapter expands upon the previous one by building upon the *Ring-a-ring-a-roses* example to explore more advanced uses of the format. Unlike the previous chapter, very few concepts are introduced because the level of discussion is now at a point where the technical descriptions become elaborate and are best reserved for the next part of this manual. Instead, the focus is on considering common desiderata and on providing examples to illustrate solutions.

Making TEI Transcriptions TAN-Compliant

Text pending.

Simplifying References and Identifying Quotations

Text pending.

Aligning Multiple Versions of Multiple Works

Text pending.

Creating Tokenization Rules

Text pending.

Analyzing Words (Morphology / Part of Speech)

Text pending.

Part II. Detailed Description

This part of the guidelines provides a detailed description of the Text Alignment Network formats. The rules for each encoding format are discussed in depth, interspersed with explanations and practical examples. This narrative, which is intended to be thorough and authoritative, is addressed primarily to those wish to read the theoretical and technical details that undergird the format. Much of this section is intended to be consulted, not read through, and it assumes that you have already read the previous part (Part I, “General Overview”).

Because readers of this section include specialists from different backgrounds, all acronyms, abbreviations, and concepts are defined and explained, even if only tersely. Specialized concepts or technologies are discussed only insofar as they affect the use of TAN; suggestions for further reading are provided.

The material in this part is organized from general to specific, then according to logical sequence (class, format, XML tree hierarchy). Much of the content follows the structure of the validation files, so both can be read in tandem.

Table of Contents

Chapter 4. General Underpinnings

This chapter retains something of the introductory spirit of the previous one by providing an overview of the fundamental principles and technologies behind TAN. The overall goal of this chapter is to summarize the theoretical and technological underpinnings that run throughout all TAN formats and to note definitions, assumptions, and other matters that broadly affect anyone working with TAN files. Although this chapter assumes no prior knowledge on your part of a particular technology, it is also not meant to be a tutorial. Links to further reading in this chapter will take you to introductory material that will provide thorough instruction or list it.

The Big Picture

The Text Alignment Network is a suite of XML encoding formats. Each TAN format is designed for a specific type of textual data, organized in three classes: segmented representations of textual works (class 1), annotations of specific texts (class 2), and rules for linguistic concepts (Class 3). The number of classes may grow in the future to cover claims about text-bearing objects, for example codicology or stemmatics (the study of how manuscripts and other versions of a work relate to each other).

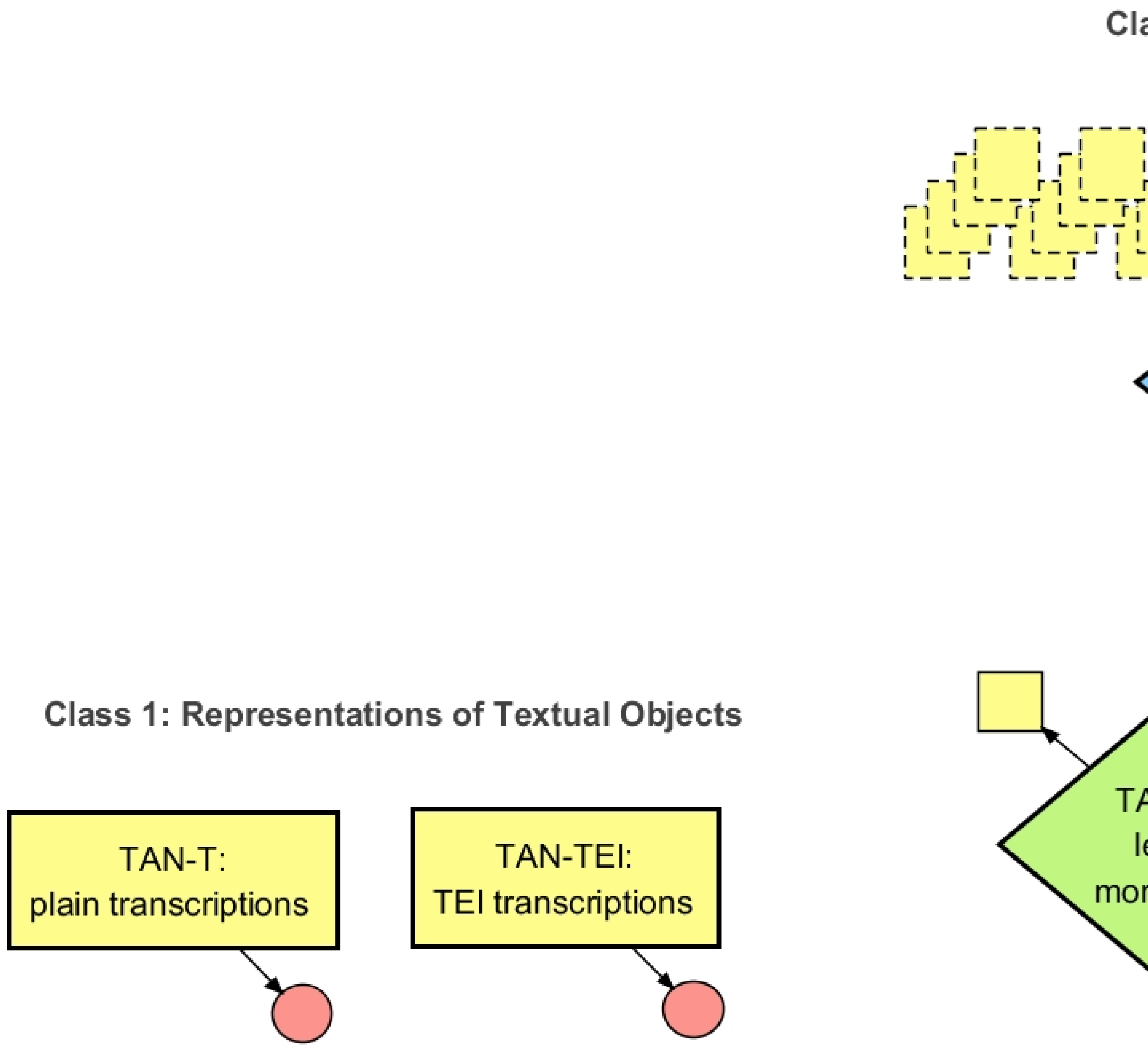
Class 1, representations of textual objects, consists solely of transcription files. Each transcription file contains the text of a single work from a single text-bearing object, whether physical or digital (TAN regards digital files as objects). There are two types of transcription file, a standard generic format and another for TEI XML files. These two types are differentiated by the name of the rootmost element in the file, `<TAN-T>` and `<TEI>` respectively. In the future, class 1 may expand to include formats intended to segment (and therefore align) visual, audio, or audiovisual files; it may also expand to include a customized form of HTML.

Class 2, annotations of textual representations, consists of alignment files and lexico-morphology files. There are two types of alignment, one for broad, general alignments and another that focuses on detailed specificity. The former, identified by the name `<TAN-A-div>` in the rootmost element, aligns any number (one or more) of class 1 files. The latter, `<TAN-A-tok>`, aligns only pairs of class 1 files. Lexico-morphology files, `<TAN-LM>`, are used to encode the lexical and morphological (or part of speech) forms of individual words in a given class 1 file. In the future, class 2 may expand to include syntax (treebanking).

Class 3, rules for linguistic concepts, consists of tokenization patterns and morphological patterns. The first of these, `<TAN-R-tok>`, declares a pattern to be used to transform a string of text into a sequence of word tokens for linguistic analysis. The second type, `<TAN-R-mor>`, is used to declare the grammatical categories or features of a given language and to stipulate rules for classifying words. Class 3 file may expand in the future to include transliteration, lexicography, and syntax.

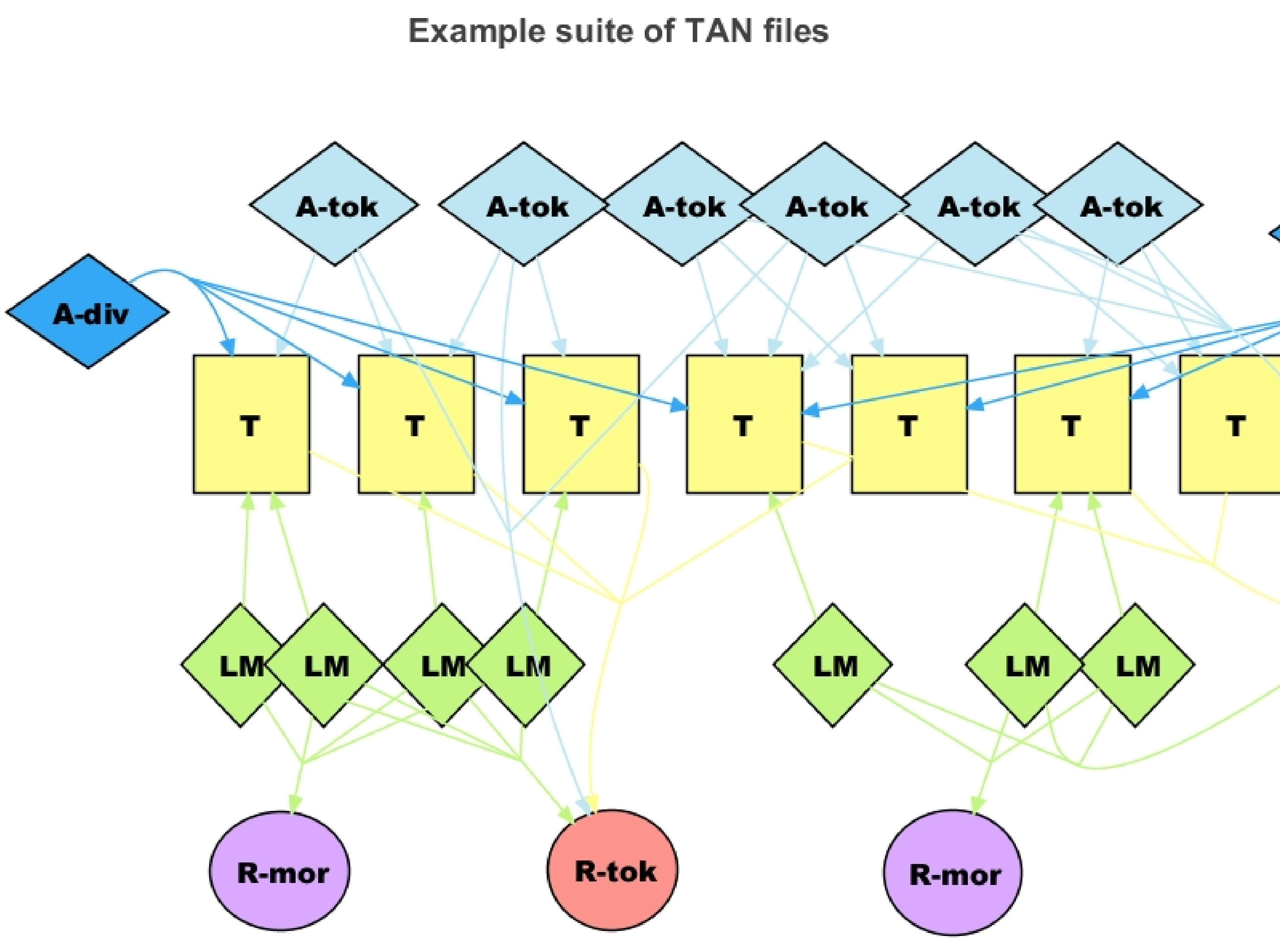
Dependence and reference: Class 1 files depend exclusively upon a single source—some text-bearing object (print or digital); they also refer to TAN-R-tok files. Class 2 files alone depend exclusively upon one or more class 1 files for their sources. They may also refer to class 3 files. Class 3 files neither depend upon nor make reference to any other class of TAN file.

Below is a visualization of each TAN format type and the types of TAN files



referenced.

TAN files that refer to or are referred to by other TAN files form a kind of network. Alignment files become the principal point of connection. Below is an illustration of a small network of TAN files.



This approach to text markup adopts what is sometimes called *stand-off annotation* (or *stand-off markup*). It is to be contrasted with *in-line annotation*, in which alignments, morphology, and so forth are built into the transcriptions themselves. Most TEI and HTML files exhibit in-line annotation. Stand-off annotation has been adopted so as to capitalize upon a number of benefits:

- An editor can work on a file with minimal distraction, focusing on a limited set of closely related questions.
- Alignments can be made concurrent to any others that may already exist, allowing for rich, complex, or alternative alignments.
- After a dataset is published, any other datasets it refers to, or any datasets referring to it, can be aggregated into much larger and more complex datasets, which can then be queried to answer questions otherwise impossible to answer.
- Editorial labor can be done without central coordination, as individuals work at their own pace, independently, on separate files.
- Errors can be corrected in a single master file. Anyone depending upon that master file as a source will be able to detect changes that have been made and deal with them accordingly.
- Any data file can be released, circulated, and used independent of any other that points to it, or to which it points.

- The TAN suite can be expanded to allow other types of linguistic data, thereby expanding the complexity of questions that could be answered.

Stand-off annotation is not without its liabilities. Dependent files might be altered or altogether deleted. Files might contain erroneous or misleading information. It can be sometimes inconvenient not to have the annotated text in the same file and place as the annotation. These liabilities mean that TAN files should not be seen as replacements for in-line annotation formats, especially those that set out to serve activities TAN was not designed to support.

Assumptions in the Creation of TAN Data

All creators and users of TAN files are expected to share few basic assumptions.

First, all TAN-compliant data is to be understood as largely *derivative*. That is, data files have no originality or creativity independent of their sources (but see below about interpretation). TAN-compliant data is to be created with intent of adhering as closely as possible to some model or archetype. For example, a transcription should replicate faithfully some earlier digital edition or text-bearing material object (e.g., stone, papyrus, manuscript, printed book for written text; audiovisual media for oral or performative texts). Morphological files and alignment files should describe as clearly and as reliably as possible their source transcriptions. *In creating and publishing a TAN file you claim to have offered a good-faith representation or description of one or more source texts or linguistic concepts; in using a TAN file, you hold the creator to that expectation.*

Second, all core TAN files are *interpretive*. That is, they are permeated by editorial assumptions and opinions that might not be shared by everyone. If there is any originality or creativity in a TAN file, it is vested in the interpretation of the sources. For example, if you edit a transcription file you must decide how to handle unusual letterforms and other visible marks. Your decisions will be informed by what you think about the original text, its native writing system, and how you interpret and use Unicode. If you write an alignment file, you must make decisions about what factors caused one text to be transformed into another. Lexicomorphological files require you to commit to one or more grammars and dictionaries, and you must discern how best to handle cases of vagueness and ambiguity. As a general rule, class 2 files have a greater interpretive dimension than do class 1 files, and class 3 than class 2. But no matter the degree, no TAN data file ever stands completely outside the interpretive act. *In creating and publishing a TAN file you claim to have disclosed the principles behind your interpretive outlook; in using a TAN file, you hold the creator to that expectation.*

Third, all core TAN files are *useful*. That is, the interpretive impluse is assumed to be coupled with an equally strong desire to make the data as useful to as wide a group of users as possible, even those who may not share your assumptions or interpretation. For example, a creator of a transcription file should normalize and segment texts with a minimum of idiosyncracies, adopting when possible reference systems that are widely known so as to optimize the alignment process. Morphological files should depend whenever possible upon grammars and lexica that are broadly used. Alignment files should work with comprehensible categories of text reuse. No TAN file will always be useful to everyone, but it should be as useful to as many as possible, as frequently as possible. *In creating a TAN file you claim to use common, shared conventions whenever possible, and to note any departures; in using a TAN file, you hold the creator to that expectation.*

There are other important assumptions that can and should be declared in a TAN file, and they are addressed in later chapters of these guidelines.

Core Technology

TAN depends upon a core set of basic, relatively stable technologies, discussed below. The technology and associated key acronyms or terms are very briefly defined and explained, to orient you to the concept, terminology, and conventions. References to further reading will lead you to better and more thorough introductions. The central goal of this section is to document and explain how certain aspects of each technology have affected the design of TAN, and therefore may significantly affect creating or processing TAN-compliant data.

Unicode

Unicode is the worldwide standard for the consistent encoding, representation, and exchange of digital texts. The standard, stable but still growing, is intended to represent all the world's writing systems, living and historical. Maintained by a nonprofit organization, Unicode is the basis upon which we can create and edit text in mixed alphabets and reliably share that data with other people, independent of specific fonts. Any Unicode-compliant text is (in general) semantically interoperable on the character level and can be exchanged between users and systems, no matter what font might be used to display the text. If some software tries to display some Unicode-compliant text in a particular font that does not support a particular alphabet, and ends up displaying boxes, the underlying data is still intact and valid. Styling the text with a font that does support the alphabet will reveal this to be the case.

With more than 110,000 characters, Unicode is almost as complex as human writing itself, and so has required a system of organization. The entire sequence of characters is divided into Unicode blocks, each one reserved, more or less, for a particular alphabet. Within each block, the various characters may be grouped further. Each character is assigned a single codepoint.

Because computers work on the binary system, it was considered ideal to number the characters or glyphs in Unicode with a related numeration system. Codepoints are therefore numbered according to a hexadecimal system (base 16), which is larger than our most common system, the decimal (base 10). The hexadecimal system uses the digits 0 through 9 and the letters A through F. (The number 10 in decimal is A in hexadecimal; decimal 11 = hex B; decimal 17 = hex 10; decimal 79 = hex 4F.) To find Unicode codepoint values is therefore helpful to think of the corpus of glyphs as a very long ribbon sixteen squares wide. This is illustrated nicely in this article [http://en.wikibooks.org/wiki/Unicode/Character_reference/0000-0FFF]. Each position along the width is labeled with a hexadecimal number (0-9, A-F) that always identifies the last digit of a character's code point value.

It is common to refer to Unicode characters by their value or their name. The value customarily starts "U+" and continues with the hexadecimal value, usually at least four digits. The official Unicode name is usually given fully in uppercase. Examples:

Table 4.1. Unicode characters

Character	Unicode value	Unicode name
" " (space)	U+0020	SPACE
®	U+00AE	REGISTERED SIGN
ю	U+044E	CYRILLIC SMALL LETTER YU

Further reading:

- Unicode Consortium [http://unicode.org]
- Unicode [http://en.wikipedia.org/wiki/Unicode] (Wikipedia)

Normalization

TAN requires all data to be normalized according to the Unicode NFC algorithm. Any text in a TAN body that does not comply will be marked as invalid. Validation engines that support Schematron Quick Fixes should provide an option to allow users to easily convert non-normalized to normalized Unicode.

Combining characters

At the core level of conformance, Unicode does not dictate whether combining characters (accents, modifying symbols) should be counted independently or as part of a base character. This affects regular expressions and XPath expressions. Any class 1 TAN file that takes a modifying letter will have a warning returned whenever it is validated. Any class 2 TAN file that attempts to count characters in a string of text that includes combining characters will be marked as invalid.

eXtensible Markup Language (XML)

The eXtensible Markup Language (XML) is a machine-actionable markup language that facilitates human readability. Defined by the W3C, XML has two versions, 1.0 and 1.1. <Q: should version 1.1 be allowed or mandated?>

At its heart, XML is rather simple. It begins with an opening line that declares that what otherwise would look just like plain text is an XML file. It then proceeds to the data, which must marked by one or more pairs of tags. An opening tag looks like `<tag>` and a closing like `</tag>` (or if the tags contain no data, this can be collapsed into one: `<tag/>`). Each pair of tags is called an element. Elements must nest within each other. They cannot overlap. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<p>A paragraph about
    <name><first>Mary</first>
        <last>Lee</last></name>.</p>
```

This nesting relationship of elements means that an XML document can be thought of as a tree or genealogy, which serves as the basis of technical vocabulary that describes the relationship between elements: root, parent, child, sibling, ancestor, and descendant. In the example above, the root element `<p>` is the parent of `<name>` and the ancestor of `<name>`, `<first>`, and `<last>`. The element `<first>` is a child of `<name>` and a descendant of both <name> and `<p>`. `<first>` and `<last>` are siblings to each other.

The opening tag of an element might have additional nodes called attributes, recognized by a word, an equals sign, and then some text within quotation marks (single or double). An element may have many attributes, and those attributes can appear in any order. Attributes can be thought of as leaves on an XML tree. They are intended to carry simple data (usually metadata about the data contained by the element), because they cannot govern a hierarchy.

```
<?xml version="1.0" encoding="UTF-8"?>
<p n="1" id="example">A paragraph about <name><first>Mary</first> <last>Lee</last>
```

The two examples above provide the same text with different line breaks, but they are functionally equivalent. That is because in most XML projects extra lines, spaces, and indentation are allowed so that a document's layout can show more clearly, through indented space, how elements nest within each other. Continuous strings of multiple spaces, tabs, and newline/carriage return are to be treated as a single space. (This behavior can be overridden in XML, but TAN files always assume that a string and its space-normalized form are equivalent.)

XML allows for other rules to be added, if an individual or group so wishes. These rules, called schemas, can allow great flexibility or be very strict. The TAN schemas tend to the latter. This is true also of its customization of the TEI schemas. For more about the strictures placed upon the TEI All schema see the section called "Transcriptions Using the Text Encoding Initiative (`<TAN-TEI>`)". See also Chapter 5, *Patterns and Structures Common to All TAN Encoding Formats* and Chapter 6, *Class 1 TAN Files, Representations of Textual Objects*.

TAN-TEI schemas are generated on the basis of the official TEI All schema that is available at the time of release.

Further reading:

- Text Encoding Initiative [http://www.tei-c.org/]

TAN and white space

<Stuff about space normalization and text nodes with nothing but blank space. The default: all whitespace text nodes will be taken as real, but will also be subject to the same kind of normalization as one gets in `fn:normalize-space()`. All leaf `<div>`s will have their space normalized. This will be applied to the concatenation of all text() descendants. Important for TEI users to know, since whitespace text nodes will be treated as a single space, and if white space appears at the beginning or end of a text node that is a descendant of a leaf `<div>`, it will not necessarily be stripped out.>

Every leaf `<div>` separated from another leaf `<div>` by one or more spacing characters is assumed to be separated by only one, but no specification is made as to whether that spacing character should be a word space (U+0020 or the XML entity &#x20;) or one of the other three possible characters (line feed, carriage return, tab). Users are expected to consult the relevant `<div-type>` to interpret the behavior. TAN is not intended to replicate the appearance of a text-bearing object. Any project that has this as a requirement should not use the TAN format.

Note

Some validation engines that process a valid TAN-compliant TEI file may return an error something like `conflicting ID-types for attribute "who" of element "comment" from namespace "tag:textalign.net,2015:ns"`. Such a message alerts you to the fact that by mixing the TEI and TAN namespaces, you open yourself up to the possibility of conflicting `xml:id` values. It is your responsibility to ensure that you have not assigned duplicate identifiers. Very often, it is possible for you to configure an XML editor to ignore this discrepancy. (In <oXygen/> XML editor go to Options > Preferences... > XML > XML Parser > RELAX NG and uncheck the box check ID/IDREF.)

Namespace

XML allow users to develop vocabularies of elements as they wish. One person may wish to use `<bank>` to refer to a financial institution, another to a river. XML was designed to allow users to mix vocabularies, even when those vocabularies use synonymous element names. This means that anyone using `<bank>` as an element name must be able to specify exactly whose vocabulary of `<bank>` is intended. Disambiguation is accomplished by associating IRIs (see the section called "International Resource Identifiers and Linked Data" below) with the element names. The actual full name of an element is the joined IRI and element name, e.g., `http://example1.com/terms/bank` and `http://example2.com/terms/bank`.

This approach to disambiguation is effective but it requires some way to associate the element name with its base IRI. If only one vocabulary is in play, then the base IRI can be declared to be the default *namespace*, which means that any subsequent element should be treated as being the *local name* of an element that is part of that namespace. The namespace is much like a surname (family name) and the local name, a given name of a person in that family. If a simple, small genealogical tree is said to belong to the Smith family, then the name "Smith" need appear only at the head, not at every person named on that tree.

But just as genalogical trees that involve other families require disambiguation, so too do XML files that mix vocabularies. But where a genealogical tree can simply add the family names at every node, XML has opted for a different mechanism, that of the *prefix*, a short string that is associated with the namespace, and is then attached to the element name.

For example, the TAN namespace is **`tag:textalign.net,2015:ns`**. The recommended prefix is tan. It is customary, then, when TAN elements are mixed with other elements such as TEI, to use the prefix to disambiguate them, e.g., `<tan:div>` versus `<tei:div>`. The fully qualified name of `<tei:div>` is properly `http://www.tei-c.org/ns/1.0/div`, whereas `<tan:div>` is `tag:textalign.net,2015:ns/div`.

Any XML element may take a special attribute (actually a pseudo-attribute, since it is technically not an attribute), `@xmlns`, sometimes on its own and sometimes followed by a colon and a prefix. The first case is used to declare the default namespace (and therefore sidestep the need to use a prefix). The second case is used to bind a namespace to a prefix. These definitions apply not only to the element that contains the `@xmlns` attribute but to all descendants, unless overridden.

Caution

There are no plans to change the namespace. But if, in the future development of the network, it becomes clear that one or more elements must be redefined in manner incompatible with previous definitions, then the namespace will be changed as well. Such a change will coincide with a major version update of the schemas (see the section called "Schemas and Validation").

The examples and the validation files illustrate the use of the TAN namespace. See especially those files pertaining to class 1 and TAN-TEI files, where the TEI and TAN vocabularies mix.

The TAN namespace uses a tag URN instead of an `http://` IRI for several reasons: (1) http namespaces rarely resolve as a URL to anything useful; (2) tag URN reinforces the lesson that the namespace is a name and not a location; (3) tag URNs are naturally constructed to allow versioning (see note above); (4) should the domain name `textalign.net` come under the ownership of someone else, the namespace will still remain valid because it is tied to a date when that domain was owned; http-prefixed namespaces are in jeopardy of being rendered invalid when a domain name changes ownership.

Schemas and Validation

XML files admit of a process called *validation*, which checks to see if all the declared rules have been followed. These validation rules are kept in files called *schemas*, plain-text files that state the rules according to one of the accepted schema languages. Each TAN file is validated by two types of schema files, one dealing with major rules concerning structure and data type (written in RELAX-NG) the other with very detailed rules (written in Schematron).

Schema files are provided in major or minor updates. An update is minor if the changes introduced do not render files valid under previous minor updates invalid. (But new warnings might be produced.) Otherwise the update will be treated as major.

Table 4.2. Locations of master schemas

Schema path	Schema files accessed
`http://textalign.net/release/1/dev/schemas/`	Major version 1, developmental version

Schema path	Schema files accessed
`http://textalign.net/release/1/schemas`	Major version 1, latest minor version

Data types

<Stuff about data types>

Data types regularly used: boolean, string, integer, IRI, date, dateTime

New paragraph introducing `@xml:lang` / xsd:language

xsd:language

TAN adopts for language identification Best Common Practices (BCP) 47, which standardizes with high precision how languages are identified. For most users of TAN, this will be a simple three-letter abbreviation, sometimes supplemented with a hyphen and a script or regional subtag. For example, `<eng>`, `<eng-UK>`, and `<eng-UK-Cyrl>` refer, respectively, to English generally, English from the United Kingdom, and English from the United Kingdom written in the Cyrillic script. As a general rule, values of this type should begin with a three-letter language code, preferably lowercase.

ISO codes for human languages appear in `@xml:lang` and `<for-lang>`. The first, an attribute, indicates the principal language of the text enclosed by the element and its descendants. But the second, an element, provides a way to make a statement about a language. For example, `<for-lang>` in the context of a TAN-R-tok file indicates languages for which a tokenization rule is appropriate.

More reading:

- BCP 47 official specifications [http://tools.ietf.org/rfc/bcp/bcp47]
- BPC 47 technical details [http://www.w3.org/TR/xmlschema11-2/#language]

<New paragraph introducing dates and times>

xsd:date, xsd:dateTime

TAN adopts the standardized ISO form of dates and times, which begins with the largest unit and moves to the smallest. A simple date takes this form: `<YYYY-MM-DD>`. A time is specified by continuing the string, first with a `<T>` (for time) then with <NEEDS TO BE WRITTEN>. Examples: <PENDING>.

The normalized value of `@when` that is most recent in a TAN document differentiates the version of a documents that share the same IRI name (see the section called "Core name, location metadata").

Future dates are not allowed in TAN files. (This is checked against the clock settings on individual computers.)

More reading:

- ISO dates

International Resource Identifiers and Linked Data

<NOT YET WRITTEN>

<Stuff about URNs and related terms *URI*, *IRI*, *URN*, *UUID*>

The network regularly depends upon three types of legally defined IRIs:

- IRI generally: https://tools.ietf.org/html/rfc3986
- UUID
- HTTP
- tag URNs:
- OID

<xml:id can't take colons; thus the IRI for TAN files has been given another attribute name, to ensure that tag URNs are always constructed validly. See below.>

Resource Description Framework (RDF) and Linked Open Data. The recommended way to prepare data for the semantic web is to coin and use URN names that take the form of URLs. The URL then allows one both to uniquely name an object and to indicate how a human or computer can get more information about it. The TAN encoding format has chosen tag URNs over URLs for several reasons:

- Permanence. Authors of TAN data are creating files that are meant to be relevant for decades and centuries in the future, well after specific domain names have changed ownership or fallen into obsolesence, and well after the creators are dead. To mint names according to URLs is inadequate for long-term use, since it has no built-in mechanism to identify who owned the domain name in question when the name was minted.
- Hassle-free URN naming. Many potential TAN authors never have owned a domain name, and never will. Further, many of those who do own domain names cannot or do not wish to configure and maintain servers to create the referral mechanisms upon which the semantic web depends. The majority of potential TAN file editors would be happy to mint names that could be later adapted for semantic web applications, but would not want to take on the extra hassle involved in writing and publishing more complicated data descriptions, e.g., RDFa.
- Disambiguation of name and location. In the semantic web, conflation of name with a location to resolve it is considered a virtue because a single string answers two questions: what is the resource and where can I find out more about it. But this conflation is unhelpful for those who use the TAN formats, who are encouraged to distribute their TAN files widely, and not rely upon a single location. And URLs are in common parlance interpreted as locations for data, not as names for things. TAN-compliant tag URLs ensure that the names of concepts and objects do not look like locations, maintaining a distinction that has always been a foundational principle in scholarly citation, namely, that one should always distinguish the name of a resource from where it might be found.

<New paragraph.>

Note

<Internationalization behind IRIs>

TAN does not constrain IRI values. Anything conforming to xsd:anyURI can be used, including formulations that have not been approved by IANA. This manual restricts itself to only approved URN schemes. <link>

IRI: http://www.ietf.org/rfc/rfc3987.txt

Tag URNs

TAN files make extensive use of the tag URN scheme. A tag URN [http://www.taguri.org] is a composite of (1) a namespace, an identifier of the authority of the TAN file and (2) a name, a string unique relative to the namespace. Those two parts, more specificially, are:

1. Namespace. `tag:` + an e-mail address or domain name owned by the person or organization that has authorized the creation of the TAN file + `,` + an arbitrary day on which that address or domain name was owned. The day is expressed in the form `<YYYY-MM-DD>`, `<YYYY-MM>`, or `<YYYY>`. A missing `<MM>` or `<DD>` is implicitly assigned the value of `<01>`.

2. Name of the TAN file. `:` + an arbitrary string (unique to the namespace chosen) chosen by the namespace owner as a label for the entire file and related versions. It need not be the same as the filename stored on a local directory. You should pick a name that is at least somewhat intelligible to human readers.

Example 4.1. TAN IRI names

```
tag:jan@example.com,1999-01-31:TAN-T001
tag:example.com,2001-04:hamlet-tan-t
tag:evagriusponticus.net,2014:tan-lm:Evagrius_Praktikos_grc_Guillaumonts
tag:bbrb@example.org,1995-04-01:pos-grc
```

The first example comes from someone who owned the email address `jan@example.com` on January 31, 1999 (at the stroke of midnight, Universal Coordinated Time). The other examples follow a similar logic. The namespace of the second and third examples are tied to the owners of specific domain names, not those of email addresses. The `2014` in the fourth example is shorthand for the first second of January 1, 2014.

Further reading:

- RFC 4151 [https://tools.ietf.org/html/rfc4151], the official definition of tag URNs

Regular Expressions

Regular expressions are patterns for searching text. The term *regular* here does not mean ordinary. Rather, it refers to rules (Latin *regula*), and points to a rule-based syntax that provides regular expressions with great power. Because regular expressions come in different flavors, and because the topic is rather complex, these guidelines are restricted to a synopsis that conforms to the definition of regular expressions found in the recommendation of XSLT 3.0 [http://www.w3.org/TR/xslt-30/#regular-expressions] (XML Schema Datatypes plus some extensions), and outlined in XPath Fuctions 3.0 [http://www.w3.org/TR/xpath-functions-30/#regex-syntax].

A regular expression search pattern is treated just like a conventional search pattern until the computer reaches a special escape character: `^ $ ( [ | . { + * ? \`

Here are a number of the special characters that are used in regular expressions, and what they mean. These tables, which are here mainly for quick reference, are not self-sufficient to ground you in regular expressions.

Table 4.3. Special characters

Symbol	Meaning
^	start of line
$	end of line
.	any character
|	or (union)
[ad]	a or d
[a-d]	a, b, c, or d
[^ad]	anything except a or d
(ad)	when ad is found treat it as a capture group (used only in a search pattern)
$1	first capture group (used only in a replacement pattern, and corresponds to the sequence of capture groups in the search pattern)

Table 4.4. Backslash special characters

Symbol	Meaning
\w	any word character
\W	any nonword character
\s	any of the four standard spacing characters: space (U+0020), tab (U+0009), newline (U+000A), carriage return (U+000D)
\S	anything not a spacing character
\d	any digit (0-9)
\D	anything not a digit
\x{hhh..}	character with the Unicode code point hhh..
\p{IsGujarati}	any character from the Unicode block named Gujarati
\\	backslash (the backslash alone suggests that the next character is a special character)
\$	dollar sign
\(	opening parenthesis
\[	opening square bracket

Table 4.5. Quantifiers

Symbol	Meaning
?	zero or one
*	zero or more
+	one or more
{n}	n times
{n,m}	from n to m times

Symbol	Meaning
[n,]	n times or more

Some examples:

Table 4.6. Examples of Regular Expressions

Expression	Meaning	Result when applied to ”Wi-fi, good. A_hem* isn’t!”
`.i`	any letter followed by i.	”Wi”, ”fi”, ” i”
`[t*].`	any t or * and the following character	”* ”, ”t!”
`\s+`	match one or more space characters	” ”, ” ”, ” ”
`\w+`	match one or more word characters	”Wi”, ”fi”, ”good”, ”A_hem”, ”isn”, ”t”
`\W+`	match one or more nonword characters	”-”, ”, ”, ”. ”, ”* ”, ”’”, ”!”
`[^q]+`	one or more characters that are not a q	”Wi-fi, good. A_hem* isn’t!”

These short examples provide a taste of how regular expressions are constructed and read. For further examples especially relevant to TAN see ???.

Regular Expressions and Combining Characters

Regular expressions come in many different flavors, and each one deals with some of the more complex issues in Unicode in their own manners. This ambiguity will be most keenly felt in the use of combining characters in Unicode. Given a string `&#x61;&#x301;&#x62;` = áb, a search pattern `a.` will in some search engines include the b and others will not.

Unicode has differentiated three levels of support for regular expressions (see official report [http://www.unicode.org/reports/tr18/]). Only level one conformance in TAN is guaranteed. If you find the need to count characters, and you are working with a language that uses combining characters, you should use token values, not numerals pointing to ordinal sequence. There is no guarantee that combining characters these will be treated uniformly. See Combining characters.

Further reading:

- Various tutorials on Regular Expressions [http://www.google.com/search?q=tutorial+regular+expressions]
- Wikipedia, Regular Expressions [http://en.wikipedia.org/wiki/Regular_expression]
- Regular Expressions in XSLT 3.0 [http://www.w3.org/TR/xslt-30/#regular-expressions]
- Unicode and Regular Expressions [http://www.unicode.org/reports/tr18/]

http://www.w3.org/TR/xmlschema-2/#regexs

XPath

<NOT YET WRITTEN>

<Stuff about XPath.>

<Examples.>

<Commonly used XPath expressions in TAN.>

<Further reading.>

Note

TAN adopts as its standard XPath 3.0.

Chapter 5. Patterns and Structures Common to All TAN Encoding Formats

Common Patterns

IRI + name Pattern

Both humans and computers will need to read and write TAN metadata. Very often what is readable to humans is unreadable to computers, and vice versa. So the TAN format requires that all metadata be provided in both forms. Although this rule may appear to introduce redundancy and therefore new opportunities for error, the clarity is critical. It is the only way at present to ensure that anyone who approaches the data--computer or human--can parse and use it. And, in reality, doubly expressed metadata provides a safeguard much like a checksum: the human- and computer-readable descriptions should correspond. Any discrepancy is a signal that an error should be diagnosed and fixed.

Some metadata, such as comments, are neither easily nor profitably translated into a computer-actionable string. In such cases only the human-readable form is required. Other metadata use regular expressions or ISO-compliant dates, both of which are well formed and usually human-legible. In those cases the human- and computer-readable components are not separated. In cases where a datum is not understandable to humans, such as a complex regular expression, a `<comment>` may be provided.

Those exceptions aside, all other metadata takes what is called the *IRI + name* pattern. (For IRI + name patterns applied to digital objects see the section called “Digital Entity Metadata Pattern”.)

Table 5.1. Synopsis of **`<IRI>`**, **`<name>`**, **`<desc>`**

name	req?	qty	parent	attributes	children	purpose
`<IRI>`	yes	one or more	varies	the section called “Edit Stamp”	data (xsd:URI)	Declare permanent, unique IRI as a name for a concept or entity
`<name>`	yes	one or more	varies	`@xml:lang`?, the section called “Edit Stamp”	data (xsd:string)	Provide human-readable name of a concept or entity
`<desc>`	no	one or more	varies	`@xml:lang`?, the section called “Edit Stamp”	data (xsd:string)	Provide human-readable description of a concept or entity

Synoptic tables

The table above is the first of many that synoptically describe the elements (or sometimes attributes) under discussion. The first column contains element names, and assumes the TAN namespace unless otherwise specified (e.g., `tei:body` would refer to the `<body>` element defined by the TEI in its namespace). The second column indicates whether the element is required by its parent. (If its parent is optional and missing then it will be missing as well.) The third column shows how many instances of the element are allowed or required. The fourth column states the element's parent and the fifth, which attributes it takes. The sixth column states what children elements or data it takes (if data, the datatype is specified in parentheses). The last column summarizes the purpose of the element. For the meaning of ?, *, etc. see Table 4.5, "Quantifiers".

Each of these tables is meant to give a short, rough overview of the elements under discussion. For an authoritative explanation of the rules and best practices, see the corresponding master schema or the prose description that follows each table.

Because `<comment>` is allowed to appear virtually everywhere, it has been dropped from the synoptic tables. For specific details on where this element is not allowed, see the section called "Comments".

One or more `<IRI>`s supply a permanent computer-readable IRI that uniquely names a concept, person, or thing. Multiple `<IRI>` values are to be treated as synonyms of the same thing, not as names of different things. In cases where the thing named is a TAN file, only one `<IRI>` is allowed.

Note

For those familiar with the Web Ontology Language [http://www.w3.org/TR/owl-ref/]: every pair of `<IRI>`s with the same parent are assumed to share the relationship `owl:sameAs`.

The element is named IRI instead of URI to encourage internationalization. Alphabets other than the Latin are welcome to be used in the construction of IRIs. For names of well-known resources, a URL identifier might be preferred (`http://...`), to facilitate linked data (see the section called "International Resource Identifiers and Linked Data"). If an entity/resource lacks a suitable URL-type name, you may use or coin any other valid IRI, such as a UUID, a tag URN, or an OID.

One or more mandatory `<name>`s furnish human-readable names of the entity identified in `<IRI>`. Because TAN files may need to be shared across language communities, this data can be stated any number of times in any language, in any Unicode-compliant script. Each human-readable datum may be given the attribute `@xml:lang`, which must take the value of a standard ISO language code (see xsd:language). Declaring the language that is used is a courtesy to users who may wish to extract explanations written in a specific language. If you provide a comment in multiple languages, it best to make sure that each version says roughly the same thing.

One or more optional `<desc>`s provide a human-readable description of the entity named in `<IRI>`. `<desc>`, which may also take an `@xml:lang` that specifies the language of the description, is in effect merely a `<comment>` that describes the entity.

All three elements above may take an edit stamp (see the section called "Edit Stamp") to specify which `<agent>` was responsible for creating or editing the data, and when.

Note

Values for `<IRI>` do not state where a source is, even though some of them may look like they do. If the value looks like a web page address (e.g., `http://...`), beware—it might be simply a name that looks like a URL, but really isn't where you will find the source. Vice versa, any `<location>` that happens fall in an IRI + name Pattern indicates only where the digital file was found; it is not to be interpreted by anyone using linked open data as the name of an entity. See the section called "International Resource Identifiers and Linked Data" and the section called "Digital Entity Metadata Pattern".

Example 5.1. IRI + name pattern: **`<role>`**

```
...........
```

```
<role xml:id="editor">
   <IRI>http://schema.org/editor</IRI>
   <name xml:lang="eng">editor</name>
</role>
   ...........
```

Example 5.2. IRI + name pattern: **`<rights-excluding-sources>`**

```
   ...........
<rights-excluding-sources rights-holder="cheh">
   <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
   <name xml:lang="eng">Creative Commons Attribution 4.0 International License.
   <desc xml:lang="eng">License granted exclusive of rights held and licenses
     held by rightsholders of the source or sources.</desc>
</rights-excluding-sources>
   ...........
```

Digital Entity Metadata Pattern

Some entities identified by the the section called "IRI + name Pattern" will be digital resources. In those cases, the IRI + name Pattern is extended in two different ways, according to whether the entity is a TAN file or not. If the entity is a TAN file, then `<IRI>` must be a valid tag URN that matches the `@id` value of the TAN file being referred to. If the entity is not a TAN file, then any IRI may be used. If you choose to use the URL of where the digital resource is as its name (as well as its location; see below), then it will be inferred that you mean to identify the digital resource that appeared at that URL at the date or time you accessed it.

Digital entities take the following extensions, in addition to their IRI + name pattern:

Table 5.2. Synopsis of **`<checksum>`**, **`<location>`**, **`<value>`**,

name	req?	qty	parent	attributes	children	purpose
`<checksum>`	no	one or more	varies	the section called "Edit Stamp"	the section called "IRI + name Pattern", `<value>`	provide the checksum of a digital entity that has been used
`<location>`	yes	one or more	varies	the section called "Edit Stamp", `@when-accessed`	the section called "IRI + name Pattern"	provide a location of a digital entity
`<value>`	yes	one	`<checksum>`	the section called "Edit Stamp"	data (xsd:string)	provide a checksum value

The optional `<checksum>` specifies some string hash function that can be used to confirm the identity of a digital file believed to be the source. The type of hash is defined in a required the section called "IRI + name Pattern", and the mandatory `<value>` provides the value of the checksum. The checksum will not be generated, checked, or validated by TAN schemas. Validation must be provided by an external tool.

One or more `<location>` elements provide URLs for where the resource can be found. Each `<location>` element must carry a `@when-accessed` value that indicates the date when the file at the URL was last consulted. If there are multiple `<location>`s, validation will be checked only against the first one available to the validator. Any `<location>` pointing to a document that is not available will marked with a warning. If no documents are available, an error message will be reported.

Note, if the digital file being referred to is a TAN file, it may have no more than one `<IRI>` in its IRI + name Pattern.

Example 5.3. Digital entity metadata pattern of a TAN file

```
      ...........
   <source xml:id="grc">
      <IRI>tag:parkj@textalign.net,2014:Evagrius_Praktikos_grc_Guillaumonts.TAN-TE
      <name>Transcription of Greek edition by the Guillaumonts</name>
      <location when-accessed="2014-10-15"
          >../TAN-T/cpg2430.grc.guillaumonts.tei.xml</location>
   </source>
      ...........
```

Example 5.4. Digital entity metadata pattern of a non-TAN file

```
      ...........
   <source>
     <IRI>tag:textalign.net,2015:source:digital:http://insaph.kcl.ac.uk/iaph2007/x
     <name>TEI transcription of boundary marker, set up by C. Julius Zoilos</name>
     <checksum>
        <IRI>http://dbpedia.org/resource/Sha-1</IRI>
        <name>SHA-1</name>
        <value>8ac1146c86e06071f352bf3cb89614c0a0f0c7ee</value>
     </checksum>
     <location when-accessed="2015-05-07"
       >http://insaph.kcl.ac.uk/iaph2007/xml/iAph010001.xml</location>
   </source>
      ...........
```

Edit Stamp

Table 5.3. Synopsis of **`@ed-who`**, **`@ed-when`**

name	req?	qty	parent	contains	purpose
`@ed-who`	yes	one	varies	`d:IDREFS`	Indicate who is responsible for edits
`@ed-when`	yes	one	varies	xsd:date, xsd:dateTime.	Indicate when edits occured

Most TAN elements allow for an optional edit stamp, an `@ed-who` and an `@ed-when` stating who created or edited the enclosed data and when. Neither attribute is allowed without the other. The value of `@when` must always conform to an ISO date or dateTime pattern. See xsd:date, xsd:dateTime.

`@ed-when`, along with `@when` (see the section called "Changes" and the section called "Comments"), are the criterion by which a TAN file's version is calculated.

An edit stamp performs the same function as `<change>`, except that no description can be provided and it points precisely to the element where a change has been made. If a description of the alteration is necessary, `<change>` should be used.

Inheritability. If without an edit stamp, an element inherits the edit stamp of its closest ancestor.

Example 5.5. Edit stamp: `<role>`

```
   ...........
<role xml:id="creator" ed-when="2015-01-29" ed-who="smith">
   <IRI>http://schema.org/creator</IRI>
   <name ed-when="2015-01-31" ed-who="cheh">creator</name>
</role>
   ...........
```

<caption>

In this example, the Edit stamp in `<role>` applies to `<role>`, `<IRI>`, and their contents, but `<name>` has its own Edit Stamp.
</caption>

Example 5.6. Edit stamp: `<change>`

```
   ...........
<change when="2015-02-04" who="cheh" ed-when="2015-02-28" ed-who="smith">Cheh c
   xml:lang values</change>
   ...........
```

<caption>

In this example, `<change>` ascribes an alteration to Cheh, but the edit stamp credits Smith with last changing the `<change>`.
</caption>

Languages

Table 5.4. Synopsis of `@xml:lang`, `<for-lang>`

name	req?	qty	parent	attributes	children	purpose
`@xml:lang`	varies	only one	varies	----	---	Identify the language in which enclosed data is written
`<for-lang>`	varies	varies	varies	the section called “Edit Stamp”	data (xsd:language)	Make a statement about a language

Languages are frequently named throughout TAN files, and always with standard ISO codes (see xsd:language). A strong distinction is made within the TAN format between a language code used to label content versus a language code used to make a statement about the language itself. The former always occurs as an attribute, the latter as an element.

Example 5.7. `@xml:lang`

```
...........
```

```
   <body xml:lang="eng">
      ...........
   </body>
...........
```

Example 5.8. **`<for-lang>`**

```
<TAN-R-tok ... >
   ...........
      <declarations>
         <for-lang>grc</for-lang>
      </declarations>
   ...........
</TAN-R-tok>
```

Overall Structure (root)

Table 5.5. Synopsis of rootmost element (**`<TAN-*>`**, **`<TEI>`**), **`@id`**, **`@TAN-version`**,**`<head>`**, and **`<body>`**

name	req?	qty	parent	attributes	children	purpose
`<TAN-*>`	yes	one	document	`@id`, `@TAN-version`, the section called "Edit Stamp"?	`<head>`, `<body>`	Indicate type of TAN format
`<tei:TEI>`	yes	one	document	`@id`, etc.*	`<tei:teiHeader>`, `<head>`, `<tei:body>`	Indicate TAN-TEI format
`/*/ @id`	yes	only one	rootmost element	----	data (tag URN)	Provide permanent, unique name for a TAN file
`/*/ @TAN-version`	yes	only one	rootmost element	----	data (xsd:string)	Name the version of validation files used
`<head>`	yes	one	`/*`	`xml:lang`?, the section called "Edit Stamp"?	`<name>`+, `<desc>`*, `<master-location>`+, `<rights-excluding-sources>`, `<source>`+, `<see-also>`*, `<declarations>`, `<agent>`+, `<role>`+, `<change>`+, `<comment>`*	Contains metadata
`<body>`	yes	one	`/*`	varies	varies	Contains data

All TAN-compliant files, no matter the type or class, follow a common basic structure: (1) at least three processing instruction nodes, (2) a namespace node, and (3) a rootmost element that has only two children: `<head>` and `<body>` (with one exception, noted below).

Processing instruction nodes: The first of three required processing nodes is the standard declaration made in every XML file's prolog: `<?xml version="1.0" encoding="UTF-8"?>` After that come two more processing instruction nodes specifying the two schema files required for validation

- `<?xml-model href="[PATH]/[ROOTMOST-ELEMENT-NAME].rnc" type="application/relax-ng-compact-syntax"?>`
- `<?xml-model href="[PATH]/[ROOTMOST-ELEMENT-NAME].sch" type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron"?>`

The first processing instruction node points to the RELAX-NG schema that declares the major rules of the chosen TAN format; the second points to the finely tuned rules, written in Schematron. Both processing instructions are required. `[PATH]` represents the pathname to the schema file, whether local or in the master location (see the section called "Schemas and Validation") and `[ROOTMOST-ELEMENT-NAME]` stands for the name of the rootmost element. Master files are housed at the TAN website, and these files may be cached, saved, served, and used anywhere. Other processing instruction nodes may be included, but conflicts may result.

Note

Of each pair of validating schema files, the first is written in RELAX-NG (compact syntax), the second in Schematron. The one semi-exception is the TAN-TEI format, which has at its core a special format developed by the TEI for customization called ODD ("One Document Does" it all; for details see the section called "Transcriptions Using the Text Encoding Initiative (`<TAN-TEI>`)"). TAN uses an ODD file that constrains the latest version of TEI All to the requirements of the TAN-T format.

Under the license under which materials are released, these schema files may be freely modified and reused. Bear in mind that any alterations run the risk of producing files that are not interoperable with TAN-compliant files.

Namespace node: All TAN elements take the namespace `tag:textalign.net,2015:ns` (subject to change; see the section called "Namespace"). In most cases, this value is placed in the rootmost element. (The only exception are TAN-TEI transcription files, which take as a default namespace `http://www.tei-c.org/ns/1.0` everywhere but in `/TEI/head`, which takes the TAN namespace.) For more about namespaces, see the section called "Schemas and Validation".

Rootmost element: The name of the rootmost element identifies the type of TAN file:

Table 5.6. Rootmost TAN elements

Rootmost element name	Type of data	TAN class
`<TAN-T>`	plain text transcriptions	1
`<TEI>`	TEI transcriptions	1
`<TAN-A-tok>`	token-based alignments	2
`<TAN-A-div>`	division-based alignments	2
`<TAN-LM>`	lexico-morphological analysis	2
`<TAN-R-mor>`	part of speech / morphology patterns	3
`<TAN-R-tok>`	tokenization patterns	3

Each rootmost element takes a mandatory `@id` and `@TAN-version`.

The value of `@id`, termed the *IRI name*, is a tag URN (no other IRI schemes are allowed; see Tag URNs for syntax).

Note

Tag URNs are used as the IRI name for TAN files for a number of reasons, foremost of which are their suitability for enduring naming schemes. That is, they will remain valid centuries from now, long after the death of the owner of a domain names or email address. It is essential to familiarize yourself with this simple but powerful IRI system. See Tag URNs.

The IRI name, which is valid independent of wherever the data may be found, is the primary way TAN files will refer to it. The namespace of the IRI name must match at least one namespace in one `<agent>`'s `<IRI>` value. This helps to tie the ownership of the TAN file to at least one person.

Great care must be taken in choosing the IRI name, because you are the sole guarantor of its uniqueness. *It is permissible for something to have multiple IRIs, but never acceptable for a IRI to name more than one thing.* It is a good practice to keep a master checklist of IRI names you have created. If you find yourself forgetting, or think you run the risk of creating duplicate IRI names, you should start afresh by creating a new namespace for your tag URNs. That is, if `tag:textalign.net,2015:...` seems to be overly cluttered, you may start a new set of names with something else, e.g., `tag:textalign.net,2015-01-02:...`.

In choosing a value of `@id` you might borrow the filename (the name one sees when one lists files in a directory), but you do not have to. Indeed, it may not be a good idea. One often needs to have the flexibility to rename files. That renaming should rarely if ever happen with TAN files that have been shared or published, because others will be citing it by the IRI name. There may be good reasons on occasion in the course of revision to create a new IRI name. Discernment is required. If you think it important dissociate a file with previous versions, only then should you change it. If you take someone else's data and alter it then you *must* change at least the namespace of the IRI name. If you do not, you falsely suggest that the owner of that namespace is responsible for the revised file. In these cases, you should supply a `<see-also>` element in the <head> indicating that you are providing an alternative or new edition. See the section called "Sources and related files" below.

The IRI name points to an edition but not a version. The specific version of a TAN file is identified by the `@when` or `@ed-when` that has the latest date. It is important, therefore, whenever you change a TAN file that has already been published to provide at least an edit stamp (the section called "Edit Stamp") in the part of the file you changed or in a `<comment>` or `<change>`. In so doing, when anyone editing a TAN file dependent upon yours validates the file, they will get a warning that changes have been made. The user may then either continue to process the file (if the changes are minor, there may be no problem) or investigate the changes before deciding what to do.

Because the entire IRI name is a single string, it is suitable for encoding formats outside of TAN (e.g., RDFa, JSON-LD, linked open data; see above, the section called "International Resource Identifiers and Linked Data").

The IRI name kept at `@id` is the only metadatum positioned outside the `<head>` element. It is placed as rootward in the document as possible to emphasize that it is the name for the entire document tree, not a specific branch.

`@TAN-version` must be the digit `1`, indicating the major version, optionally followed by the minor version number. If the minor version is missing, the latest version is assumed. These version numbers must match the schemas invoked in the prolog or the file well be rendered invalid. For versions of schemas, see the section called "Schemas and Validation".

The rootmost element takes only two children: `<head>` and `<body>`, the latter containing data and the former, metadata (data about the data). The only exception to this rule are TAN-TEI files, which take three children: `<teiHeader>`, `<head>`, and `<text>`. (See Chapter 6, *Class 1 TAN Files, Representations of Textual Objects*.) Thus, most every TAN file generally looks something like this:

Example 5.9. Common Structure of a TAN file

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-A.rnc" type="appl
<?xml-model href="http://textalign.net/release/1/dev/schemas/TAN-A.sch" type="appl
<TAN-A-div xmlns="tag:textalign.net,2015:ns" TAN-version="1" id="tag:parkj@textali
   <head>
      ...........
   </head>
   <body>
      ...........
   </body>
</TAN-A>
```

Metadata (<head>)

No matter how much one TAN format differs from another, the metadata are quite similar. Anyone getting a TAN file, no matter its class or type, is assumed to want to know, and therefore find easily and predictably, the following:

1. the name;
2. the version;
3. the sources;
4. other files that have an important relationship;
5. the most significant parts of the editorial history;
6. the linguistic or scholarly conventions that have been adopted in creating the data;
7. the license, i.e., who holds what rights to the data, and what kind of reuse is allowed.
8. the persons, organizations, or entities that helped create the data, and the roles played by each.

To answer these questions completely, consistently, and predictably across all TAN formats, the `<head>`, a mandatory child of the rootmost element, takes a common pattern, thus allowing anyone (people or computers) to work easily and predictably across large numbers and types of TAN files. The TAN `<head>`, intended to be concise and focused, compels you to provide metadata for the data that is governed by `<body>`, but it does not require metadata for the metadata. That is, your metadata should focus on the data itself and not other things. For example, `<head>` requires you name the people who helped create or edit the data, but you are not expected to give those person's birthdays, nationality, etc. You merely refer through `<IRI>` to other authoritative sources that can provide more information about that person.

Core name, location metadata

Table 5.7. Synopsis of **`<name>`**, **`<desc>`**, **`<master-location>`**

name	req?	qty	parent	attributes	children	purpose
`<name>`	yes	one or more	`<head>`	`xml:lang`?	data (xsd:string)	Provide human-readable name of a TAN file
`<desc>`	no	one or more	`<head>`	`xml:lang`?	data (xsd:string)	Provide human-readable description of a TAN file

name	req?	qty	parent	attributes	children	purpose
`<master-location>`	yes	one or more	`<head>`	----	data (xsd:anyURI)	Provide location of where a master of a TAN file can be found

As was mentioned above (see the section called “Overall Structure (root)”) the computer-readable form of the name is kept at `@id` in the rootmost element. The human-readable counterpart is one or more `<name>`s, the first child of `<head>`. `<name>` can contain any string. Some editors may wish to use the filename; others, a descriptive phrase. The decision is left entirely to you. An `@xml:lang` may be used to hold the value of the language of the content of `<name>`. Multiple `<name>` elements are to be treated as synonyms.

One or more optional `<desc>` elements may be provided, to describe the TAN file more fully.

One or more `<master-location>`s may be provided, to indicate where master versions of the file is to be found. It takes as a value any URI, and may be absolute (e.g., `http://...`) or relative. Any relative URI will be resolved according to the parent directory of the current TAN document. If a document in a `<master-location>` cannot be found, a warning will be returned. All copies of a TAN file found through `<master-location>` should have the same version; if they are not, a warning will be returned from the validation process.

When you use `<master-location>` you commit yourself to updating the TAN file in those locations. Similarly, if you have completed work on a TAN file, you must commit yourself to a `<master-location>`. If `@in-progress` (see the section called “@in-progress”) is false and there is no `<master-location>` the document will be reported as invalid.

Example 5.10. Name metadata

```
<TAN-T xmlns="tag:textalign.net,2015:ns"
   id="tag:jan@example.com,1999-01-31:TAN-T001" TAN-version="1">
   <head>
      <name>Smith poem 1</name>
      <desc xml:lang="eng">J. Smith's poem no. 1, TAN-compliant.</desc>
      <master-location>smithpoem1.xml</location>
      ...........
   </head>
   ...........
</TAN-T>
```

Rights and Licenses

Table 5.8. Synopsis of **`<rights-excluding-sources>`**

name	req?	qty	parent	attributes	children	purpose
`<rights-excluding-sources>`	yes	one	`<head>`	the section called “Edit Stamp”?, `@rights-holder`	the section called “IRI + name Pattern”	declare intellectual property rights associated with data, exclusive of the rights held over the sources

Under `<rights-excluding-sources>`, a mandatory child element of `<head>`, you indicate the license under which the data is released. The value of `@rights-holder` (required) points to one or more `<agent>`s (see the section called “Agents”) who hold the rights over the data, *exclusive* of any rights held over the sources. This element contains the the section called “IRI + name Pattern”.

To reduce confusion on the parts of users of your TAN data, you may not use more than one license (that is, there must be one and only one instance of `<rights-excluding-sources>`). If parts of your TAN data fall under two or more licenses, you should (1) split the file (one for each license), (2) find a license that sythesizes the desired licenses, or (3) release the entire file under the most restrictive of the possible licenses.

Diligently check to ensure that the license you have claimed respects the rights of your sources' rightsholders. It is recommended that you license your data under a license that is similar to or slightly more liberal than the one under which your sources have been released.

Whereas `<rights-excluding-sources>` is mandatory in every TAN file, `<rights-source-only>` is not (see below). It is never allowed in class 2 files, because the sources are always TAN files, and the rights information can be drawn from those sources. For class 1 and class 3 files, declaring the rightsholders of your source(s) may be more difficult, perhaps even risky, and is therefore optional.

As an editor, you are strongly encouraged in the `<desc>` element of `<rights-excluding-sources>` to emphasize the distinction between the rights you have over your data and the rights held by others over your source. A statement something like this is recommended: `<desc>The data in this file, only insofar as it constitutes an independent work, is licensed exclusive of any licenses held by parties over the source or sources listed below.</desc>`

License Discrepancies

If you find a discrepancy between the two licenses--that of a TAN file and that of its sources--the more restrictive license should be respected. In using a TAN file, it is your responsibility to investigate the entire chain of rights. If a TAN user declares a very liberal, open license for the data, this does not necessarily mean that the material upon which it depends is in the public domain. The source may be under tightly regulated rights.

Example 5.11. License metadata: data only

```
   ...........
<rights-excluding-sources rights-holder="park">
   <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
   <name>Creative Commons Attribution 4.0 International License</name>
   <desc>Exclusive of rights held and licenses offered by rightsholders of the
      sources listed below, this data file, only insofar as it constitutes an i
      work, is licensed under a Creative Commons Attribution 4.0 International
</rights-excluding-sources>
   ...........
```

Sources and related files

Table 5.9. Synopsis of **`<source>`**, **`<rights-source-only>`**, **`<see-also>`**, **`<relationship>`**

name	req?	qty	parent	attributes	children	purpose
`<source>`	yes	varies*	`<head>`	the section called "Edit	the section called "IRI + name Pattern",	state name of source(s) upon which the data depends

name	req?	qty	parent	attributes	children	purpose
				Stamp"?, `@xml:id`?	the section called "Digital Entity Metadata Pattern"?**, `<rights-source-only>`*	
`<rights-source-only>`	no	one	`<source>`	the section called "Edit Stamp"?, `@src`?, `@rights-holder`?	the section called "IRI + name Pattern"	declare intellectual property rights over a source
`<see-also>`	no	one or more	`<head>`	the section called "Edit Stamp"?	the section called "IRI + name Pattern", the section called "Digital Entity Metadata Pattern"?, `<type>`	identify digital file that relates somehow to the current file
`<relationship>`	yes	one	`<auxiliary>`	the section called "Edit Stamp"?	keyword `old version`, `new version`, `dependent`, `auxiliary` or the section called "IRI + name Pattern"	indicate what relationship the referred file holds to the context file

Note

* `<source>` quantity is either exactly one (TAN-T, TAN-TEI, TAN-LM), exactly two (TAN-A-tok), two or more (TAN-A-div), or one or more (TAN-R-mor).

** Digital Entity Metadata is required of all class 2 files, and of all TAN files that depend upon a digital source.

The mandatory `<source>` names the item or items upon which the TAN data depends. This element contains the pattern IRI + name (see the section called "IRI + name Pattern"). It allows `@xml:id` only for TAN formats that permit more than one `<source>`.

Transcription files (TAN-T[EI]) will have as their source some digital file or text-bearing material object. It is wise to find a URN that other people might use as well, for example a minted URL that reliably names the object (see examples below). By supplying `<IRI>` with a name in the form of an `http` URL provided by a library catalogue, you provide a way for users to retrieve extensive, structured bibliographical information. You then save yourself the hassle of writing a detailed bibliographical description and your users the hassle of parsing and importing it into their reference management system. If a URL cannot be found for `<IRI>` (perhaps the text-bearing object is a papyrus or a T-shirt), you may simply coin a tag URN or a UUID. Alternatively, if you find another

TAN file that uses the same source, it would be a good idea to adopt that name. (You don't need to ask permission to use names minted in other TAN files; all URNs are in the public domain.)

Class 2 files (lexico-morphology and alignment files) will always have as their source one or more class 1 files (transcriptions). In those cases the `<IRI>` element must give each TAN-T file's IRI name (the value of `TAN-T/@id`; see the section called "Core name, location metadata" above). The Digital Entity Metadata pattern is mandatory because the sources are necessarily digital.

Class 3 TAN files (i.e., TAN-R-mor, TAN-R-tok) have more complex `<source>` needs. These are discussed in Chapter 8, *Class 3 TAN Files: Language and Script*.

If you can determine with authority the rights of your sources (see below), you may use `<rights-source-only>`s, which take the IRI + name pattern plus an optional `@rights-holder` to point to the `<agent>`s that hold the rights over the source. This element in class 3 files take an optional `@src` attribute, pointing by means of id references of one or more `<source>`s to which the statement applies. There may not be more `<rights-source-only>`s than `<source>`s.

Naming Licenses and Rights Holders for Sources

It is recommended that you not declare who own what rights over your source unless you yourself are the rightsholder or unless you can make this assertion with certainty. Copyright laws differ from one country to another, and they have changed over time. A source may be protected by copyright in one place and simultaneously be in the public domain in another. (For example, neither the U.S. nor Canada have explicit provisions about the copyright status of modern editions of ancient texts; Germany, however, explicitly allows forty years of copyright protection before the text goes into the public domain.) Some copyright statements in books are false, or cannot be proven. Some persons or entities who claim rights over a source may have no legal basis for the claim, at least in some jurisdictions. Furthermore, if you mischaracterize the rights that are held over a source, you may be held liable by a putative rights holder. It is safer to use the `<IRI>` of `<source>` (described below) to point the user to authorities that have greater authority and specificity about who owns what rights.

A missing `@rights-holder` implies only only silence, not that the rightsholder is unknown or nonexistent. If you know that a source is in the public domain, it is recommended that you use the IRI `http://creativecommons.org/publicdomain/mark/1.0/` and omit `@rights-holder`. If you know the rights holder of a source has waived all rights and placed the work in the public domain, you may use the IRI `https://creativecommons.org/publicdomain/zero/1.0/`.

Copyright Law versus Contract Law

Some third-party services, such as the Thesaurus Linguae Graecae for Greek texts, require users to agree not to copy and reuse the texts in service's databases. Such agreements fall under the area of contract law and not copyright law. That is, these third parties are most often not the putative rights holder for the texts they store, therefore, they should normally not be credited in any `<rights-source-only>`. See also Digital Transcriptions: Credit as Source or Not?

The optional `<see-also>` is reserved for cases where the editing of a TAN data file depends upon other textual objects you wish to credit or refer to. Any number of these can be provided. They take one `<relationship>` in conjunction with an IRI + name pattern.

`<relationship>` takes either a reserved keyword or the IRI + name pattern, to define a relationship not covered by the keywords. The reserved keywords:

Table 5.10. **`<relationship>`** reserved keywords

keyword	equivalent IRI	description
`alternative edition`	`tag:textalign.net,2015:relationship:alternative-edition`	Alternative edition of the present file.
`old edition`	`tag:textalign.net,2015:relationship:old-edition`	TAN file superceded by the present file.
`new edition`	`tag:textalign.net,2015:relationship:new-edition`	TAN file that supercedes the present file.
`dependent`	`tag:textalign.net,2015:relationship:dependent`	TAN file that uses the present file as a source.
`auxiliary`	`tag:textalign.net,2015:relationship:auxiliary`	Digital or nondigital entity that was helpful in creating or editing the present file.

`alternative edition`: Used when one wishes to refer to another TAN file that provides essentially the same data with a different approach. There is no assumption that this edition is necessarily better. For class 1 files, this option may be used to point to an edition of the transcription that uses a different reference system for segmenting and labeling the text. For class 2 files, to alterantives that use largely the same sources or works, but with different declarative assumptions (e.g., tokenization). Class 3 alternative editions might be similar rules for the same languages.

`old edition`: Used to point to a TAN file superceded by the present one. The alterations have been significant enough to warrant a new `@id` in the rootmost element and you believe the current file is superior.

`new edition`: Used to point to a TAN file that supercedes the present one. The alterations have been significant enough to warrant a new `@id` in the rootmost element and you believe the current file is inferior. This is useful for officially deprecating a TAN file without deleting it. If it is used as a source by other TAN files, when the latter are validated a warning will be returned.

`dependent`: Used to point to a TAN file that uses the current file as a source.

`auxiliary`: points to an item, whether physical or digital, that assisted in creating or editing the current file. This may be useful for cases where you begin with an item as a source, but in the course of editing you find a better source, but still want to credit the intermediary item.

If these keywords are insufficient, you may define your own relationship by specifing it through the IRI + name pattern.

Digital Transcriptions: Credit as Source or Not?

A third-party website or database may have a digital form of a transcription that you wish to use as the basis for a TAN-T or TAN-TEI file. These digital surrogates are versions in their own right. They almost always depart from their original source (print or digital) even if in small ways. Should they be listed under `<source>` or `<see-also>`?

If you are interested in and focused on the digital transcription itself, then that should be credited and treated as a `<source>`. But many digital transcriptions are simply derivative copies or versions of other sources, whether print or digital. It is almost always better to go up the chain of dependence until you find the earliest version that still meets your needs. Make that your `<source>`. You can then make the third-party digital transcription a `<see-also>`. In crediting the third party, you do not assign them any rights. But beware, lest any agreement you have made puts you in violation of contract law. See above, Copyright Law versus Contract Law.

Example 5.12. **`<source>`**

```
   ...........
<source>
   <IRI>http://www.worldcat.org/oclc/14388933</IRI>
   <name>Wang, Fuzhi, 1619-1692. Zhonghua shu ju : Xin hua
           shu dian Beijing fa xing suo fa xing, Di 1 ban. 1962.</name>
</source>
   ...........
```

Example 5.13. **`<source>`**, **`<rights-source-only>`**, and **`<see-also>`**

```
   ...........
<source>
   <IRI>http://lccn.loc.gov/72309768</IRI>
   <name xml:lang="fra">Traitè pratique, ou, Le moine. Evagre le Pontique.
      Introduction par Antoine Guillaumont et Claire Guillaumont. Paris,
      Éditions du Cerf, 1971.</name>
   <rights-source-only>
      <IRI>urn:uuid:d56dd4a9-5487-4131-ac61-f43e2b11a17a</IRI>
      <name>Exclusive rights</name>
      <desc>Rights for the source are held by Sources chrétiennes, Les Éditions
         des Tanneries 75013 Paris France www.editionsducerf.fr</desc>
   </rights-source-only>
</source>
<see-also>
   <relationship>auxiliary</relationship>
   <IRI>tag:parkj@textalign.net,2014:Dysinger_website_Praktikos#2014-02-02</IRI
   <name>Dysinger's handtyped Greek, accessed 2014-02-02</name>
   <type>
      <IRI>tag:textalign.net,2015:auxiliary:basetext</IRI>
      <name>Initial, base digital text used as the starting point for an
      accurate rendition of the source</name>
   </type>
</auxiliary>
   ...........
```

Declarations

Table 5.11. Synopsis of **`<declarations>`**

name	req?	qty	parent	attributes	children	purpose
`<declarations>`	yes	one	`<head>`	----	varies	collect declarations of scholarly assumptions and decisions made in preparing data

Each core TAN data file has exactly one `<declaration>` element. The structure and content of this element differs from one type of TAN file to the next, so detailed discussion is reserved for later chapters.

Agents

Table 5.12. Synopsis of `<agent>`

name	req?	qty	parent	attributes	children	purpose
`<agent>`	yes	one or more	`<head>`	`@roles`?, `@xml:id`	the section called “IRI + name Pattern”	identifies persons, organizations, or artificial intelligences used to prepare the data

One or more `<agent>` elements must be included as children of `<head>`, to describe people, organizations, or other entities that created or edited the data, in whole or part, directly or indirectly. At least one `<agent>` must have an `<IRI>` that is a tag URN whose namespace matches that of the IRI name. The first `<agent>` with an `<IRI>` that has a substring identical to the namespace of the IRI name is, by default, taken to describe the person or organization that is authorized the creation of the TAN file and is ultimately responsible for it. If no such substring match can be made, then the file is invalid.

`<agent>` may describe any human or organization, or an entity that simulates a human or organization (i.e. `<agent>` may name an artificial intelligence). `<agent>` takes the the section called “IRI + name Pattern”.

Many people involved with a project will not have a name in the form of a URL; in those cases, a minted UUID or tag URN will suffice as the value of `<IRI>`.

Each `<agent>` element must take an `@xml:id` whose value is unique to the document. This value may be used throughout the TAN file to assign responsibility for a role, comment, or change to a particular person or organization.

Each `<agent>` element may include `@roles`, which carries a space-delimited list of identifying references to `<role>`s (see the section called “Roles”). If you need to provide a more complex account of how various agents helped create the the file, you should consider ignoring this attribute and using `<agentrole>` instead (see the section called “Agent roles”).

Example 5.14. `<agent>`

```
   ...........
<agent xml:id="kalvesmaki" roles="publisher editor">
   <IRI>http://viaf.org/viaf/299582703</IRI>
   <name xml:lang="eng">Joel Kalvesmaki</name>
</agent>
<agent xml:id="example" roles="editor">
   <IRI>urn:uuid:64befba0-a0a6-11e3-8975-0002a5d5c51b</IRI>
   <IRI>http://example.org/24929</IRI>
   <name xml:lang="eng">Bill Contributor</name>
</agent>
<agent xml:id="example2" roles="caterer">
   <IRI>tag:jan@example.org,2014:my-cook</IRI>
   <name xml:lang="eng">Robin the Chef, who fed the caterer and editor.</name>
</agent>
   ...........
```

Roles

Table 5.13. Synopsis of **`<role>`**

name	req?	qty	parent	attributes	children	purpose
`<role>`	yes	one or more	`<head>`	`@xml:id`	the section called "IRI + name Pattern"	identifies responsibilities or activities assumed in preparing the data

One or more `<role>`s must be included as children of `<head>`, identifying tasks and responsibilities of agents involved in creating and editing the data. A role may be any activity, e.g., editor, funder, supervisor, data-processor, peer reviewer, patron. These roles may be chosen however you see fit, and are declared through the the section called "IRI + name Pattern".

Each `<role>` must have an `@xml:id` with an value unique to the document to be used as cross-references in `<agent>` or `<agentrole>` (see the section called "Agents" or the section called "Agent roles").

Roles will be defined quite differently from one project or culture to another. People can reasonably disagree on terminology or definition. You should put in `<IRI>` values that you believe define those roles with the level of precision or vagueness you think most appropriate. If you have a very special definition of a job, you may wish to coin your own IRI and provide an exact definition in `<desc>`. Or if you have chosen a URL for an IRI, you may wish to define it more precisely in a Linked Data approach. To draw from well-established vocabularies, see iris.xml.

Note that it is not necessary to declare a role of rights holder. This relationship is already built into the `@rights-holder` (must point to one or more `<agent>`s) in `<rights-excluding-sources>` and `<rights-source-only>`. See the section called "Rights and Licenses".

Example 5.15. **`<role>`**

```
    ...........
<role xml:id="editor">
    <IRI>http://schema.org/editor</IRI>
    <name xml:lang="eng">Editor</name>
</role>
<role xml:id="publisher">
    <IRI>http://schema.org/publisher</IRI>
    <name xml:lang="eng">Publisher</name>
</role>
<role xml:id="caterer">
    <IRI>urn:uuid:0cab862e-986f-4cd7-ac12-963cd5359c1b</IRI>
    <name xml:lang="eng">Caterer (preparer and donator of food)</name>
</role>
    ...........
```

Agent roles

Table 5.14. Synopsis of **`<agentrole>`**

name	req?	qty	parent	attributes	children	purpose
`<agentrole>`	no	one or more	`<head>`	`@when-from`, `@when-to`, `@agents`,	----	specifies a time period when one or more

name	req?	qty	parent	attributes	children	purpose
				`@roles`, the section called "Edit Stamp"?		agents held one or more roles

To document a complex history of a TAN file, `<head>` may have one or more `<agentrole>` elements. `<agentrole>`, an empty element, takes four required attributes: `@when-from`, `@when-to`, `@agents`, and `@roles`. The last two attributes carry identifying references to the appropriate `<agent>` and `<role>` elements, respectively (multiple values should be space delimited). The other two attributes contain ISO-compliant time values indicating when the agent(s) had that role(s).

Keep in mind that credit (or blame) for specific parts of the file will be based upon how you combine `<role>` and `<agent>` elements, so be certain that persons and their responsibilities are accurately described. Be sure to name and credit the person or organization that owns the namespace part of the IRI name (see above). Be sure to credit everyone involved in creating and editing your data.

Example 5.16. **`<agentrole>`**

```
  ...........
<agentrole when-from="2013-12-31" when-to="2014-01-21"
   who="johnson smith vanBuren" roles="editor"/>
   ...........
```

Changes

Table 5.15. Synopsis of **`<change>`**

name	req?	qty	parent	attributes	children	purpose
`<change>`	yes	one or more	`<head>`	`@who`, `@when`, the section called "Edit Stamp"	data (xsd:string)	specifies a change made to a TAN file

The element `<head>` must have one or more `<change>` children. Collectively, `<change>` elements form the changelog, the revision history of the document. The content of each `<change>` element is expected to be human-readable only. Each change element must have `@when` and `@who` values documenting the time the change was posted and the `<agent>`s that made the change (multiple values should be space delimited). `@when` must point to a time or a date + time (see xsd:date, xsd:dateTime) and `@who` must refer to an `<agent>` (multiple values should be space delimited). An edit stamp may be provided to specify who recorded the change.

It is up to you to decide how many `<change>`s you should record and collect. Ideally, `<change>`s documenting every published version should be retained.

`<change>` elements may appear in any order, but it is good practice to put the most recent at the top.

Example 5.17. **`<change>`**

```
   ...........
<change when="2014-02-07" who="park">Corrected word in p1s1.</change>
<change when="2014-02-07" who="cheh">Logged changes today on
   behalf of all agents.</change>
<change when="2014-02-02" who="park">Eliminated line breaks, hyphenations.</cha
<change when="2014-01-24" who="park">Began new file, based upon predecessor mod
   urn:uuid:85c5a1cc-51a1-11e3-822c-ce3f5508acd9::2014-01-16 </change>
```

```
          ...........
```

Comments

Table 5.16. Synopsis of **`<comment>`**

name	req?	qty	parent	attributes	children	purpose
`<comment>`	no	one or more	varies	`@who`, `@when`	data (xsd:string)	specify any comment

Although discussed here as the last child of `<head>`, `<comment>`s may placed anywhere in a TAN document except for the `<body>` of class 1 files. Each `<comment>` must have `@when` and `@who` values, documenting the time the comment was made and the `<agent>`s that made it. `@when` is part of the calculus to determine the version of a particular TAN file. See the section called "Edit Stamp".

Example 5.18. **`<comment>`**

```
   ...........
<comment when="2014-02-02" who="example">This text still needs to be
   proofread carefully!</comment>
<comment when="2014-01-24" who="park">This document was prepared
   as a sample text for the Text Alignment Network (TAN-T).</comment>
   ...........
```

Data (<body>)

Regardless of class, each TAN format is devoted to a particular type of textual information, and therefore requires its own structure. A transcription, for example, will be structured quite unlike a set of lexico-morphology data. Thus, `/*/body`, unlike `/*/head`, for the most part differs widely from one format to another, and is discussed in specific chapters. But some common features are documented here.

@in-progress

In every TAN file, `<body>` takes an optional `@in-progress`. If missing, this value is assumed to be true. That is, the TAN file is a work in progress. When you finish any TAN file, be certain to explicitly declare `in-progress="false"` before publishing it. You must also be sure to include at least one `<master-location>` under `<head>` indicating where master copies of the TAN file are kept.

Any TAN file marked implicitly or explicitly as being in progress means only that the editor either has not yet finished supplying the data or reserves the right to make major changes.

Caution

If you find a TAN file that is marked as not being in progress, this is not to suggest that the file is error free. Any editor of a TAN file reserves the right to correct typographical errors or mistakes.

Chapter 6. Class 1 TAN Files, Representations of Textual Objects

Files in class 1 formats are segmented representations of textual objects. This class consists solely of transcriptions, the foundation of any TAN-compliant project. No class 2 files (e.g., alignment, morphology) can be created without class 1 files.

Transcriptions come in two different formats, identified by the rootmost element. `<TAN-T>` is a simple, generic format, as close as one can get to plain text. `<TEI>` (also referred to in this manual as TAN-TEI), on the other hand, is highly expressive. Because the two types function almost identically, the generic format is first presented in full, and is followed by a second part devoted to special features necessary for TEI.

The different encoding formats are illustrated throughout this and the next two chapters with two examples, a simple one, *Ring a Ring o' Roses* (see Chapter 2, *Starting off with the TAN Format*), and a more complicated one, the *Praktikos* by Evagrius of Pontus.

Where necessary, supplements from other examples will be provided to illustrate advanced features of the encoding rules.

Principles and Assumptions

To maximize the utility of transcriptions within the TAN ecosystem, certain assumptions and recommendations are made regarding class 1 (transcription) files. These assumptions about transcriptions complement the more general ones; see the section called "Design Principles".

First, TAN transcription files are restricted exclusively to the creation of an accurate digital representation of a single version of a single work, correctly segmented and labeled and drawn from only one source. Editors of TAN-T or TAN-TEI files should be able to read, write, and proofread texts in the languages of the sources. They should understand the works well enough to segment them and label them. They should be familiar with the normalizing conventions adopted by scholars who work on texts from the same time, language, and culture. They should have some familiarity with how users of the text will want to segment the transcription on the basis of words.

Editors need not understand everything about their texts, such as where and how they are derived or the history of the work. They need not have any specialized skill in grammar or lexicology. They need not know the morphology of individual words, or how individual parts of the text have been translated. Those skills are reserved for other core TAN formats.

TAN-T(EI) editors stand at the beginning of a larger workflow for text alignment. It is critical that work not be published hastily, and only after careful proofreading, especially of white-space, which will significantly affect later use, especially in processes involving tokenization Many transcriptions, especially those of long texts, have typographical errors. But eliminating as many as possible before publication will maximize the utility of a TAN-T(EI) file.

If you are creating a TAN-T(EI) file, you are doing so primarily to service text alignment. To align is to correlate passages of text that attest to acts of text reuse, e.g., translating, paraphrasing, revising, quoting, and summarizing. Common to all these processes is that one or more texts, usually called the *source* (or *sources*), are transformed into a new text, oftentimes called the *target*. A target is based upon the semantics of the source and not its visible features. That is, translators, whether working literally or freely, move from idea to idea, clause to clause, sentence to sentence, and word to word. Line and page breaks in a source usually leave negligible if any trace in the target. Therefore a

TAN transcription file should not try to reflect the appearance of its source (i.e., it should not be a diplomatic edition), and it should be structured, when possible, by the most familiar reference system for that work. If possible, semantic mileposts (clauses, sentences, paragraphs, chapters) not visual (lines, columns, pages, volumes) should be preferred. (But even alternative reference systems for the same text can be reconciled in the TAN format.)

Contributors and users of TAN files also assume a fundamental distinction between a textual object (or a text-bearing object) and a conceptual work. The former has materiality and the latter does not. Even though both are constitutively necessary for any transcription, the two must always be differentiated. For example, when we refer in ordinary conversation to the *Odyssey* we refer to an idea, concept, or model, but not to a particular copy, version, or edition. On the other hand, if we encounter a book with *The Odyssey* on the cover, we do not necessarily assume that it meets our ideal. It might contain only parts of it, or it might include other works. In ordinary conversation, we usually indicate whether our use of "Odyssey" refers to the work or to a specific copy.

Both the conceptual and physical categories are critical constraints a TAN-T(EI) file. In brief: *Every TAN-T(EI) file must be restricted to a transcription of a single version of a single conceptual work found on a single text-bearing object, segmented and labeled according to a single reference system.* This restrictive principle is critical to the the success of the network, because it is the basis upon which other TAN files will be aligned. Adherence to the principle reduces the risk of confusion, simplifies the files, and allows complexity to be shifted from an individual file to the network in which that file participates.

Single textual object. Each TAN-T(EI) file transcribes one and only one object. It may be a digital file or a text-bearing artefact—a particular book, manuscript, stone, label, etc. If the object you've chosen is virtually indistinguishable from other related objects (e.g., copies of a printed book or copies of a digital file), then the entire class of mechanically reproduced objects is to be treated as a single object (an entity some librarians call a manifestation). Some borderline cases about what constitutes an object will require discernment and judgment. For example, some manuscripts have been split up, their parts now residing in multiple libraries around the world; other manuscripts have been physically altered or supplemented. In such cases, you may need to define your single text-bearing object in a way that might not conform strictly to how others might define it. You have the right and responsibility to define your object in the way that makes most sense to you and your users.

Single conceptual work. Identifying and defining the creative work you mean to transcribe is your decision. Suppose the text-bearing object you have is a Bible. It can take whatever contours you wish. For example, given that a text-bearing object is a New Testament, you may wish to define the entirety as a single work, or you may wish to make the work the Gospel of Matthew, or a specific episode in that gospel, or simply the Beatitudes. Any reasonable definition of a work is permitted, but a TAN-T(EI) file must contain nothing but the work that has been chosen. A TAN-T(EI) file need not contain the entire work. But it should be complete, and preserve the order found in the text-bearing object.

Single version. If the text-bearing object repeats the work in multiple versions (different languages or different editions), you must choose one and only one. If you wish to transcribe the other versions in your project, they should take separate TAN-T(EI) files. If you wish to work a transcription that is a critical edition or a collation of multiple versions of a text then it is best to prepare and publish separately a print or digital edition of that edition. That new edition can then be treated as the single version for your TAN-T file.

Note

Transcriptions of unwritten texts. The term "textual object" includes physical or digital media that preserve written, aural, or performative text, e.g., a speech delivered in Russian or a dialogue performed in American Sign Language. A TAN-T(EI) file may directly transcribe an oral or performative text, as long as it adheres as faithfully as possible to the source, and makes a full declaration of all normalizations that have been exercised. In such cases a list might be quite extensive and unwieldy, so it may be preferable to create and publish an independent transcription to then serve as the source of a TAN-T(EI) file.

Domain model. The categories used in this introduction may remind some readers of the domain model defined by the Functional Requirements for Bibliographical Records (FRBR), which identifies four types of Class 1 entities: *Work*, *Expression*, *Manifestation*, and *Item*, the first pair being conceptual, non-material entities and the latter pair material ones. The TAN encoding format is built upon a slightly different perspective. FRBR Items are equivalent to what TAN calls *objects* (a term chosen to emphasize their materiality). Multiple objects that for all intents and purposes are indistinguishable (i.e., items reproduced mechanically) are equivalent to FRBR Manifestations, but in TAN no separate entity has been defined. It is best to think of TAN objects as being equivalent to FRBR Items, with FRBR Manifestations being sets of indistinguishable TAN objects. As for conceptual entities, the network has been designed with the assumption that most TAN users will find the distinction between Works and Expressions to be unhelpful or false. What one person calls a FRBR Expression another may legitimately call a Work. TAN assumes that any derivation of a Work (or Works) is itself a Work, which is really shorthand for *work-version*. Thus, in this manual the term *version* indicates not an entity separate from work, but rather a work that is known either to derive from another or to be the basis for other derivations. TAN avoids altogether the term *Expression*, which necessarily implies a medium (without which nothing can be expressed) and therefore materiality.

The above explanation is, for the moment, provisional. The concepts behind the TAN conceptual model may be more precisely defined in the future.

Single reference system. Every TAN transcription must be segmented into a hierarchy of uniquely labeled divisions. Those divisions, whenever possible, should align first and foremost according to the reference system that is most familiar when the text is cited across different versions, a so-called canonical reference system. (Because the canonicity of a reference system admits degrees and dispute, the simpler term *reference system* is preferred and applies to all such systems, whether "canonical" or not.) Preference should be shown to the natural semantic divisions of the work, not the physical features of a particular object. Chapter, paragraph, and sentence numbers are preferable to volume, page, and line numbers, because other versions of the work (other translations, paraphrases) will only roughly, if at all, follow an object-oriented reference system. But if a given work has multiple systems (e.g., the works of Plato and Aristotle, which have both semantic- and object-oriented referenc systems), there are ways that both can be represented (explained below <XREF pending>).

If there is a good reference system, but the divisions are overly lengthy, you may introduce subdivisions. If there is no reference system, or if you think that the ones that exist are inadequate or misguided, you may create one of your own. If you develop your own reference system, be sure to make it useful and consistent, not just for your version but for any others of the same work. All references must be unique, and they must conform to a hierarchical model. Like XML itself, a reference system must follow a treelike hierarchy: one-to-many relationships from the root element down, with no overlapping siblings.

The previous paragraphs describe segmentation that usually applies to phrases, sentences, paragraphs, and similarly sized portions of text. But you must also prepare your files for token- or word-based segmentation. It is critical that you think about how to normalize the text to optimize tokenization (see the section called "Tokenization Patterns").

Judicious normalization is critical, especially of white space. You should be familiar with the conventions that are commonly used for identifying word tokens in the languages of your text. Some languages are relatively straightforward and word tokens can be isolated at word spaces. But in some languages word tokens might be joined with other word tokens, without any spaces. In these cases, the major words should be delimited by some consistent character, usually standardized white space, and the component words or morphemes should be delimited by nonstandardized white space. In many cases U+200B (ZERO-WIDTH SPACE) is appropriate as a delimitery within a string of characters to separate words. It is recommended that the spelled out XML entity be used (i.e., `&#x200B;`), so that editors and readers of the text can easily see where these markers have been set.

Other types of normalization that asist analysis and segmentation should also be undertaken. This usually entails the suppression of such elements extraneous to the text such as parenthetical editorial insertions, stray handwritten remarks, discretionary word-breaking hyphens, editorial comments, inserted cross-references, and reference numerals (page numbers, section numbers, etc.). In addition, you may wish to resolve ligatures and to correct unintended typographical errors. In a digital source, precise spacing marks you might be tempted to represent with General Punctuation (U+2000..U+200A; U+200B is a special case--see above) should be converted to ordinary spaces, and superscript combining Roman letters (U+0363..U+036F) should probably be converted to their non-combining counterparts. All Unicode must be normalized to NFC forms (see Normalization). Editors must declare what kind of tokenization their transcription has been optimized to take.

Not all decisions will be clear-cut. You may justly hesitate before normalizing orthography, punctuation, accentuation, or capitalization. Some aspects of Unicode that lend themselves to varying conventions may need special consideration. You may need to consider whether an unusual or rarely used Unicode character might be misinterpreted, or a hindrance to other users (especially for parsing word tokens). Describe any decisions that might not be agreeable to everyone who uses the file in the header (see the section called "Filter: Normalizing, Replacing, Transliterating").

You should remove from the text anything that is not part of the work proper and would interfere with detailed word-for-word alignment, or would create many extra hours of preprocessing or postprocessing work for later users. If your source is an electronic text that has preserved line breaks, you may need to ensure that the sibling elements that split the single word have no space (including line breaks) in any intervening text node. By default, XML treats multiple instances of space characters (space, tab, newline, and carriage return) as a single space. If retention of multiple spaces is important for your research, then the TAN formats are not an appropriate XML format, because it relies upon space normalization.

General Transcriptions (`<TAN-T>`)

The rootmost element of a TAN-T file is `<TAN-T>`.

`head/declarations`

The TAN-T `<head>` is identical to all other TAN heads except for the `<declarations>` element.

Table 6.1. Synopsis of TAN-T **`<declaration>`**

name	req?	qty	parent	attributes	children	purpose
`<declarations>`	yes	one	`<head>`	the section called "Edit Stamp"?	`<work>`, `<version>`?, `<div-type>`+, `<recommended-tokenization>`+, `<filter>`?	identifies the work that has been transcribed

The TAN-T `<declarations>` takes five possible child elements—three mandatory, two optional. These declarations pertain to the work and version chosen, the types of segementation, the tokenization patterns adopted, and normalizations made to the transcription.

Works

Table 6.2. Synopsis of **`<work>`**

name	req?	qty	parent	attributes	children	purpose
`<work>`	yes	one	`<declaration>`		the section called "IRI + name Pattern"	identifies the work that has been transcribed

The `<work>` element is required, and states which conceptual work has been transcribed. It takes the the section called "IRI + name Pattern".

Well-known works may have a suitable IRI name already assigned to them. See iris.xml for suggestions on vocabularies. Most works have not been assigned IRIs or are named in IRI vocabularies that are not well known. You may assign any work your own URN, through a UUID or a tag URN. Any IRIs that you mint are free to be used by other people writing TAN files about the same work. Similarly, if you find that another TAN-T file has transcribed a version of your work, you may also use that URN (you don't need to ask permission, since no URN can be copyrighted). As with other parts of the metadata, multiple `<IRI>`s and `<name>`s are synonyms for the same work, not individual names for different works.

Example 6.1. **`<work>`**

```
   ...........
<work>
    <IRI>http://dbpedia.org/resource/Ring_a_Ring_o%27_Roses</IRI>
    <name>"Ring a Ring o' Roses" or "Ring Around the Rosie"</name>
</work>
   ...........
```

Example 6.2. **`<work>`**

```
   ...........
<work>
   <IRI>http://dbpedia.org/resource/Praktikos</IRI>
   <IRI>tag:evagriusponticus.net,2014:cpg2032</IRI>
   <name xml:lang="eng">The Praktikos, written by Evagrius Ponticus</name>
</work>
   ...........
```

Versions

Table 6.3. Synopsis of **`<version>`**

name	req?	qty	parent	attributes	children	purpose
`<version>`	no	one	`<declaration>`		the section called "IRI + name Pattern"	identifies the work-version that has been transcribed

An optional `<version>` element identifies a specific version of what has been identified under `<work>`. It takes the the section called "IRI + name Pattern".

In most cases, `<version>` is unnecessary, because `<work>` in conjunction with `<source>` are sufficient to identify a particular work-version. But if the source carries multiple versions (e.g., a bilingual edition of a text), then `<version>` must be included.

Very few work-versions have their own URN names. It is advisable to assign a tag URN or a UUID. If you have used an IRI for `<work>` that you are entitled to modify, you may wish to add a suffix that will name the version. If you need to specify exactly where on a text-bearing object a version appears, `<desc>` or `<comment>` should be used.

Example 6.3. **`<version>`**

```
      ...........
   <version>
      <IRI>urn:uuid:31648039-3dbb-49b9-b66e-9bd2cd11630e</IRI>
      <name>zweite Version des Arbeits</name>
   </version>
      ...........
```

Example 6.4. **`<version>`**

```
      ...........
   <version>
      <IRI>tag:evagriusponticus.net,2014:cpg2040.grc.1971</IRI>
      <name xml:lang="eng">Guillaumonts' ancient Greek version of the
          Praktikos published in 1971</name>
   </version>
      ...........
```

Division Types

Table 6.4. Synopsis of **`<div-type>`**

name	req?	qty	parent	attributes	children	purpose
`<div-type>`	yes	one or more	`<declaration>`	`@xml:id`, `@ns-are-numerals`?	the section called “IRI + name Pattern”	identifies the types of division into which the transcription has been segmented

At least one `<div-type>` must be supplied, declaring the types of divisions into which the text has been segmented. Each `<div-type>` must take an `xml:id` that has a string value (regular expression: `\w+`; no punctuation or spaces are allowed) unique to the document. This value will be used throughout the TAN file to label divisions, so it is advisable to keep the abbreviation brief but meaningful. Each `<div-type>` takes the the section called “IRI + name Pattern”.

`@ns-are-numerals` is optional. If absent, the value is assumed to be true, and processors of the TAN-T file will attempt to cast the values of `@n` as `xs:integer`.

TAN supports four numeration systems other than Arabic numerals:

1. Roman numerals. Values up to 5000, utilizing i, v, x, l, c, d, and m, uppercase or lowercase, with liberal syntactic rules (within a roman numeral, any digit preceding one of a higher value is assumed to be a subtraction from the total value; all others are positive values).

2. Alphabetic sequences. The 26-letter Roman alphabet, assigned 1 through 26, with each series 26n + 1 beginning with n + 1 repeated letters. E.g., aa = 27, bbb = 54. Uppercase or lowercase allowed.

3. Arabic numerals + alphabetic sequences. Arabic numerals followed immediately by an alphabetic sequence. The second item is to be calculated as a subsequence of the first item, with the lack of a second item taking highest priority. E.g., 4, 4a, 4b, 4c....

4. Alphabetic sequences + Arabic numerals: As above, but with alphabetic sequence preceding Arabic numerals.

Implementations processing TAN files should determine whether the `@n`s are numerals by testing all values and opting for the numeration system that predominates. It is recommended that the functions and variables defined for TAN-A-div [../functions/TAN-A-div-functions.xsl], TAN-class-2 [../functions/TAN-class-2-functions.xsl], and TAN-core [../functions/TAN-core-functions.xsl] be studied in reconciling and converting numeration systems.

In most cases, `@ns-are-numerals` will not need to be invoked. Even if there is a case where `n="civ"` and `civ` is meant to be a word or abbreviation, this will be interpreted as a Roman numeral only if the majority of `@n`s for that `<div>` type match the pattern for Roman numerals. In that case, `@ns-are-numerals="false"` should be invoked in the appropriate `<div-type>`.

Well-known division types already have suitable IRI names. See iris.xml for suggestions. That list also includes less common types of divisions. If you encounter a rare division type, or one that needs specificity not provided for in a well-known URN, you should mint your own.

You may have as many `<div-type>`s as you like. Not every division type needs to be used in the transcription. In this it resembles surplus namespace invocations at the head of an XML document. The only harm is bloat.

Example 6.5. `<div-type>`

```
   ...........
<div-type xml:id="line">
   <IRI>http://dbpedia.org/resource/Line_(poetry)</IRI>
   <name>line of poetry</name>
</div-type>
   ...........
```

Example 6.6. `<div-type>`

```
   ...........
<div-type xml:id="p">
   <IRI>http://dbpedia.org/resource/Preface</IRI>
   <name xml:lang="eng">preface</name>
</div-type>
<div-type xml:id="t">
   <IRI>http://dbpedia.org/resource/Title_(publishing)</IRI>
   <name xml:lang="eng">title</name>
</div-type>
<div-type xml:id="s">
   <IRI>http://dbpedia.org/resource/Section_(typography)</IRI>
   <name xml:lang="eng">section</name>
</div-type>
<div-type xml:id="c">
   <IRI>http://dbpedia.org/resource/Chapter_(books)</IRI>
```

```
      <name xml:lang="eng">chapter</name>
      <name xml:lang="grc">#########</name>
   </div-type>
   <div-type xml:id="e">
      <IRI>http://dbpedia.org/resource/Epilogue</IRI>
      <name xml:lang="eng">epilogue</name>
   </div-type>
      ...........
```

Recommended Tokenizations

Table 6.5. Synopsis of **`<recommended-tokenization>`**

name	req?	qty	parent	attributes	children	purpose
`<recommended-tokenization>`	yes	one or more	`<declarations>`	the section called "Edit Stamp"?, `@xml:id`, `@which`?	the section called "Digital Entity Metadata Pattern"?	names a TAN-R-tok file that is recommended

At least one `<recommended-tokenization>` must be supplied, declaring which TAN-R-tok patterns the transcription has been optimized for (see the section called "Tokenization Patterns"). If no thought has been given to tokenization, this element should be left empty. Otherwise this element takes two kinds of content:

1. Core TAN tokenization pattern. TAN provides three core tokenization patterns, `general-1`, `general-words-only-1`, and `precise-1`. To refer to these core patterns, simply assign to `@which` one of those three values. The numeral after the name refers to version number; the number of versions may grow. See the section called "Tokenization Patterns" for more about these files.

2. Custom TAN tokenization pattern. Under this option, Each `<recommended-tokenization>` must take a unique value for `@xml:id` that names the tokenization pattern. This value will be used by class 2 files to choose a particular tokenizing pattern. Each `<recommended-tokenization>` takes the the section called "Digital Entity Metadata Pattern".

If a TAN-R-tok file is language specific, and if not every language invoked in a TAN-T(EI) file is explicitly supported by that tokenization file, a warning will be produced. If you have in your transcription combining characters that cannot be handled reliably by a given recommended tokenization pattern, you will get an error upon validation, indicating where the combining characters are to be found. For more see Regular Expressions and Combining Characters.

You may have as many `<recommended-tokenization>`s as you like. The order of `<recommended-tokenization>`s is important. The most recommended tokenizing pattern should be placed first.

Example 6.7. **`<recommended-tokenization>`**: none

```
      ...........
   <recommended-tokenization/>
   ...........
```

Example 6.8. **`<recommended-tokenization>`**: all three core general TAN tokenization pattern

```
   ...........
<recommended-tokenization which="general-1"/>
<recommended-tokenization which="general-words-only-1"/>
<recommended-tokenization which="precise-1"/>
   ...........
```

<caption>

The example above prioritizes the core TAN general pattern above the other two.
</caption>

Example 6.9. **`<recommended-tokenization>`**: modern English texts

```
   ...........
<recommended-tokenization xml:id="penn">
   <IRI>tag:textalign.net,2015:tokenization:eng.1</IRI>
   <name>Tokenization pattern for English, in Penn Treebank style</name>
   <location when-accessed="2015-03-17">../TAN-R-tok/tok.eng.1.xml</location>
</recommended-tokenization>
<recommended-tokenization which="general"/>
   ...........
```

Filter: Normalizing, Replacing, Transliterating

The `<filter>` of a TAN-T file takes three children, all optional and repeatable: `<normalization>`, `<replace>`, and `<transliteration>`.

Table 6.6. Synopsis of **`<filter>`**, **`<normalization>`**, **`<replace>`**, **`<transliteration>`**, **`<pattern>`**, **`<replacement>`**, **`<flags>`**

name	req?	qty	parent	attributes	children	purpose
`<filter>`	no	one	`<declarations>`	@src?	`<normalization>`*, `<replace>`*, `<transliteration>`*	declare textual changes that have been made to the source
`<normalization>`	no	one or more	`<filter>`	@xml:id?	`<for-lang>`*, the section called “IRI + name Pattern”	declare normalizations made to the text of a source
`<replace>`	no	one or more	`<filter>`	@xpath?	`<pattern>`, `<replacement>`, `<flags>`?	state a pattern to be searched and replaced in a digital source
`<transliteration>`	no	one or more	`<filter>`	---	`<for-lang>`*, the	state a transliteration scheme used upon a

name	req?	qty	parent	attributes	children	purpose
					section called “IRI + name Pattern”	source to convert one alphabet to another
`<pattern>`	yes	one	`<replace>`		data (xsd:string)	state through a regular expression a pattern to be found
`<replacement>`	yes	one	`<replace>`		data (xsd:string)	state through a regular expression a replacement pattern
`<flags>`	yes	one	`<replace>`		data (xsd:string)	state flags to be applied in a replace operation

`<filter>` (see the section called “Filter: Normalizing, Replacing, Transliterating” above) may take take one or more `<normalization>`s, each of which states an editorial decision made to normalize the text found in a source. Each normalization takes the the section called “IRI + name Pattern” and may take one or more `<for-lang>` values, indicating through a language code to which languages the normalization applies (see the section called “Languages” and xsd:language).

`<normalization>` caters to descriptions of changes to nondigital sources, but it may be made also for digital sources, to declare global changes that would be cumbersome, difficult, or impossible to describe in `<replace>`.

In creating a TAN-T file you should declare every change you have made to the source, no matter its medium. See the section called “Principles and Assumptions” above. Currently there are no IRI vocabularies for normalizing decisions; see iris.xml for TAN-minted ones. If none of these adequately describe a normalization principle you have adopted, you may mint your own IRI, probably a tag URN. It is hoped that as TAN-T files are created, commonly used normalizations will coalesce into a shared IRI vocabulary.

`<replace>`s, intended exclusively for valid XML sources, is used to indicate what global changes were applied to the text of the source before creating the data.

The conventions of `<replace>` accord with `fn:replace` as defined in XPath Functions 3.0 [http://www.w3.org/TR/xpath-functions-30/#func-replace]. Each `<replace>` has as children exactly one `<pattern>`, exactly one `<replacement>`, and an optional `<flags>` element, corresponding to the three parameters in the XPath function. The values of these three elements, all strings, are regular expressions (see the section called “Regular Expressions”). Also, if the source is a valid XML file, `<replace>` may take an optional `@xpath` value, to restrict the scope of the replacement to a specific place in a valid XML source.

Caution

Using `@xpath` is not recommended if you wish to serve users who may be relying upon XSLT 2.0 and earlier for transformations. Only from version 3.0 has it been possible to convert a string datatype (the datatype of `@xpath`) into an XPath expression.

Caution

It is not predictable whether a regular expression engine will include or exclude combining characters in a search. You will need to worry about this only if you are working with texts that use combining characters and you are making subtoken alignments. For more details see the section called "Regular Expressions" and Combining characters. If you invoke a TAN-R-tok file that has a replace function, and the target text includes combining characters, a warning will be returned upon validation.

Multiple `<replace>` values are assumed to have been applied in the order given.

In the third part of `<filter>` are one or more optional `<transliteration>` elements, which indicate what if any transliteration schemes have been used upon the source (digital or nondigital). This may be useful for projects that find it easier to work with a source in a Latin alphabet rather than a native one. Each transliteration element takes the the section called "IRI + name Pattern" and one or more optional `<for-lang>` values, indicating through a language code to which languages the transliteration applies (see the section called "Languages" and xsd:language).

Caution

It is hoped that a class 3 format will be developed to formalize transliteration rules. Until this format is available, there is no standardized way to convert the transliteration. For now, the `<IRI>`s in `<transliteration>` should use vocabularies similar to those recommended in iris.xml.

Example 6.10. **`<normalization>`**

```
      ...........
   <filter>
      <normalization>
         <IRI>tag:textalign.net,2015:normalization:comments-editorial-removed</IRI
         <name>Editorial comments removed.</name>
      </normalization>
   </filter>
      ...........
```

Example 6.11. **`<normalization>`** and **`<replace>`**: remove discretionary hyphenation and line breaks from a TAN-TEI file

```
      ...........
   <filter>
      <normalization>
         <for-lang>grc</for-lang>
         <IRI>tag:textalign.net,2015:normalization:hyphens-discretionary-removed</
         <name>Suppression of hyphens</name>
      </normalization>
      <replace>
         <pattern>&lt;lb[^/]*\s*break\s*=\s*"no"[^/]*\s*/></pattern>
         <replacement/>
      </replace>
      <replace>
         <pattern>&lt;lb[^/]*\s*[^/]*/></pattern>
         <replacement> </replacement>
      </replace>
   </filter>
      ...........
```

<caption>

The removal of discretionary hyphens is declared with `<normalization>` and not `<replace>` because the TAN-T editor has realized that some hyphens at line ends are part of the word. A generalized regular expression cannot be constructed to define this change. The first `<replace>` declares that all instances of TEI's `<lb>` that are meant to suppress word breaks should be deleted. The regular expression held by `<pattern>` tells the computer to look for any instance of `<lb` followed by anything until it reaches `break="no"` then to keep going until it finds `/>`, even across new lines. Note the use of `&lt;`, since a literal use of `<` would render this file as invalid. The second replace replaces all remaining TEI `<lb>` with a single space.
</caption>

Example 6.12. **`<transliteration>`**

```
   ...........
<filter>
   <transliteration>
      <for-lang>grc</for-lang>
      <IRI>tag:textalign.net,2015:transliteration:grc:lat:betacode2013</IRI>
      <name>Romanized Greek</name>
   </transliteration>
</filter>
   ...........
```

Data

Table 6.7. Synopsis of TAN-T **`<body>`**, **`<div>`**

name	req?	qty	parent	attributes	children	purpose
`<body>`	yes	one	`/TAN-T`	`@xml:lang`, the section called "@in-progress"?	`<div>`+	contain a transcription
`<div>`	yes	one or more	`<body>` | `<div>`	`@type`, `@n`, `@xml:lang`?	`<div>`+ | text	segment, structure, and name the segments of a transcription

The sole purpose of `<body>` is to contain a segmented transcription of a single version of a single work from a textual object. `<body>` may take `@in-progress` (see the section called "@in-progress").

`<body>` takes one or more `<div>` elements each of which governs zero or more other `<div>` elements. Collectively, this tree of `<div>` elements (with `<body>` at the root) is used to structure the text according to the reference system that has been adopted. Furthermore, only leaf `<div>` elements, that is, `<div>` elements that contain no other `<div>`s (XPath expression `//tan:div[not(descendant::tan:div)]`) are allowed to contain text children (white space excepted). That is, any `<div>` will contain one of the two following combinations of children:

1. one or more `<div>`s and nothing else (except white space)

2. no `<div>`s at all and text (or TEI-compliant markup, if a TAN-TEI file).

`@xml:lang`, which is required in `<body>` and is optional in any `<div>`, indicates the default language of the portions of the transcription governed by the parent of `@xml:lang`. This value is inherited by all descendants, unless overridden by a new `@xml:lang`. That is, each `<div>` takes the `@xml:lang` value of itself or its nearest ancestor. Any time a transcription moves from one language to another (and the difference is obvious and substantial, not merely a word or short phrase), you should indicate this with a new `@xml:lang` in the appropriate `<div>`. This is a courtesy to users who may be looking for text from a particular language. If a change in language occurs within a leaf `<div>`, you should ensure that the `@xml:lang` value of that `<div>` (explicity or by inheritance) reflects the majority language. Alternatively, you may subdivide that `<div>`s into a new set of leaf `<div>`s, with the appropriate `@xml:lang` values applied.

Each `<div>` must take a `@type`, which points to the `@xml:id` values of a `<div-type>`.

Each `<div>` must include `@n`, which carries a label that identifies the number or name of a segment of the transcription. Although defined as a string, `@n` will be cast by processors as an `xs:integer` if possible, and if this feature has not been overridden. See the section called “Division Types”.

Within this treelike structure of `<div>`s, any given `<div>` will be identified by the paired values of `@type` and `@n`, going from the most ancestral `<div>` down to the context node. The concatenation of the entire chain is called the *flat ref*, and it serves as an identifying device. One of the most important validation rules must here be noted, the *Leaf Div Uniqueness Rule*, which states that the flat ref for each leaf `<div>` must be unique.

Flattened References, and the Leaf Div Uniqueness Rule

Each leaf `<div>` must have a full reference that is unique within the document. Take for example a work that is structured into books, chapters, and finally sections. If you are told to find book 3, chapter 7, section b, line 5, you should be able to find the one place where that applies. There should not be two passages labeled 3.7.b.5.

It is helpful to visualize the way the leaf `<div>` uniqueness rule gets calculated, because this process is used to validate class 2 files. First, imagine each `<div>` in the hierarchy having their `@n` and `@div` values joined by a single space or punctuation character to form a single string, e.g., `book.3`, `chapter.7`, `section.b`, and `line.5`. Then picture the entire hierarchy of `<div>`s collapsing into a flat list of leaf `<div>`s. Once collapsed, each leaf `<div>` takes a new value that combines the reference values of its ancestors, joining them once again by some space or punctuation mark, e.g., `book.3:chapter.7:section.b:line.5`. (The choice of periods and colons is entirely arbitrary; see Punctuation in flattened references)

Each flat ref must be unique. Any transcription file that has leaf `<div>`s with duplicate flattened references will be rendered invalid.

This rule applies only to leaf `<div>`s and not to `<div>`s in general. That is because there are many cases where a source will begin a major textual unit, interject another one, then resume the previous one. For example, chapters 24 and 30 in the book of Proverbs of the Septuagint are split in half and collated. The sequence goes as follows: 24.1 - 24.22e (22a - 22e are verses not extant in the Hebrew); 30.1 - 30.14; 24.23 - 24.34; and 30.15 - 30.33. If the uniqueness rule applied universally, this kind of reference system would be impossible.

If a section of text traditionally carries no label, for example an unnumbered preface, this must declared explicitly with an empty value, i.e., `@n=""`. In such cases look out for ambigua: any other sibling `<div>` of the same type that would assign its `@n` as a null value (for example, two prefaces to the same work).

Example 6.13. TAN-T **`<body>`**, **`<div>`**

```
...........
<body xml:lang="eng" in-progress="false">
   <div type="line" n="1">Ring-a-ring-a-roses,</div>
   <div type="line" n="2">A pocket full of posies;</div>
   <div type="line" n="3">Hush! Hush! Hush! Hush!</div>
   <div type="line" n="4">We're all tumbled down.</div>
</body>
...........
```

Transcriptions Using the Text Encoding Initiative (`<TAN-TEI>`)

Note

This section is to be read in conjunction with the section called "General Transcriptions (`<TAN-T>`)" and the section called "eXtensible Markup Language (XML)", which addresses some technical issues that relate to TAN-compliant TEI to XML and validation generally.

Some creators and editors of transcriptions will find the rather stripped-down TAN-T format inadequate. Some may wish to mark up the text in leaf `<div>`s. Others may already have a library of transcriptions with detailed annotations that are desirable to keep, even if TAN users may not be interested. To serve these needs, TAN has adopted the Text Encoding Intiative (TEI) format for its expressiveness, its stability, its flexibility, and its widespread use in scholarship.

TEI is one of the best known XML encoding formats for textual markup. It was designed to be maximally expressive and flexible, to serve the most robust needs of humanities scholars. In serving this mission, TEI has come to define more than five hundred different element names, and more than two hundred attributes. Of course, any given TEI file uses only a small subset of those elements and attributes, and TEI itself comes in different sizes, from TEI Lite, which uses only 75 attributes and 140 elements, to TEI All, which opens up almost the entire library.

The TEI has been designed to be customizable, which is actively encouraged. Individuals and projects may define their own subset of TEI elements, to constrict or expand the allowable rules as they see fit. TAN-TEI is one of those customizations, adjusting the TEI rules to ensure that the transcriptions are interchangeable and suitable with all other TAN files.

The customization of the TEI can be summarized as follows (the default namespace in this section is the TEI namespace, `http://www.tei-c.org/ns/1.0`):

Table 6.8. Synopsis of TAN-TEI customization

TEI element	summary of alteration
`<TEI>`	• must have `@id` with IRI name • should take new namespace declaration, `xmlns:tan="tag:textalign.net,2015:ns"` • takes a new child element, `<head>`, placed between `<teiHeader>` and `<text>`
`<body>`	• must take `@xml:lang` • may take `@in-progress` • must take exclusively one or more `<div>`s • overall contents must be restricted to a single work

TEI element	summary of alteration
`<div>`	• must take either only `<div>`s or no `<div>`s at all • must take `@type`, `@n`

Like all other TAN files, the rootmost elements of TAN-TEI files must take an `@id`, the IRI name. See above, the section called "Core name, location metadata".

The `<teiHeader>` is retained, but for TAN purposes, the contents do not matter. If your `<teiHeader>` has any kind of metadata relevant to TAN users, you will need to copy and modify the data for `<tan:head>`. You must follow the TAN rules for head (see the section called "Metadata (`<head>`)" and the TAN-T the section called "`head/declarations`"). You may find that some of the material you put in `<teiHeader>` is not expected in `<head>` and that some of the expectations of `<head>` are nowhere in your `<teiHeader>`.

The opening tage of `<tan:head>` (the sibling element between `<teiHeader>` and `<text>`) will declare the TAN namespace to be its default, i.e., `<head xmlns="tag:textalign.net,2015:ns">` or `<tan:head>` if the prefix `tan:` has been defined in the rootmost element.

The TEI `<body>` must take `@xml:lang`, and it may take `@in-progress` (the section called "@in-progress"). It must take one or `<div>`s, which, just like TAN-T files, take either only `<div>`s or no `<div>`s at all. This hierarchy of `<div>`s must be devoted to only one work. Each `<div>` must take `@type` and `@n`.

Within any leaf `<div>`, you may use whatever TEI markup you wish, to whatever level of depth or complexity. All users of your TAN-TEI file will be interested in the text nodes, but only a subset will care about any markup within leaf `<div>`s. For this reason, even if you change the value of `@xml:lang` within a leaf `<div>`, there is no guarantee that readers or processors of your data will take it into account. Or if you try to represent the physical appearance of the text on the object, it is likely to be ignored. TAN rules on normalizing space and Unicode characters also prevails over any exemptions declared in TEI.

As a matter of practicality, it is helpful to envision the TEI to TAN-TEI conversion process as falling in three steps:

1. Structure: insert new processing instructions (TAN-TEI validation files); adjust rootmost element by supplying IRI name to `@id`, TAN namespace to `@xmlns:tan`.
2. Metadata: create new `<head>` and populate with correct metadata
3. Data: edit `<body>` to restrict the content to a single work; restructure `<body>` content into nesting `<div>`s with unique `@type` and `@n` values.

It has been the experience of those who have made TEI to TAN-TEI conversions that step 2 is the most time-consuming. The TAN `<head>` requires one to more carefully curate the metadata than does `<teiHeader>`.

Example 6.14. TAN-TEI file

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="../schemas/TAN-TEI-all.rnc" type="application/relax-ng-compact-s
<?xml-model href="../schemas/TAN-T.sch" type="application/xml" schematypens="http:
<TEI n="tag:textalign.net,2015:Evagrius_Praktikos_grc_Guillaumonts.TAN-TEI.001"
   xmlns="http://www.tei-c.org/ns/1.0" xmlns:tan="tag:textalign.net,2015:ns">
   <teiHeader>
   ...........
   </teiHeader>
   <head xmlns="tag:textalign.net,2015:ns">
   ...........
   </head>
   <text>
      <body xml:lang="grc">
         <div div-type="p" n="">
            <div div-type="t" n="1" suppress-normalization="hyph">
               <pb n="482"/>
               <p>######## #######</p>
            </div>
            <div div-type="s" n="1">
               <p>######...</p>
            </div>
         ...........
         <div div-type="t" n="1">
            <pb n="496"/>
            <p>### ##### ##### ######### </p>
            <p>######## #´</p>
         </div>
         <div div-type="c" n="1">
            <p><pb n="498"/>#´</p>
            <p>#########µ## #### ###µ#...</p>
         </div>
         ...........
         <div div-type="e" n="1">
            <p>#### ###...</p>
         </div>
         ...........
      </body>
   </text>
</TEI>
```

Chapter 7. Class 2 TAN files, Annotations of Texts

This chapter is devoted to class 2 formats, all of which annotate class 1 files. Class 2 files deal with either alignment or lexico-morphology.

Alignment files come in two different formats, identified by the rootmost element. TAN-A-div is macroscopic; TAN-A-tok, microscopic. TAN-A-div aligns any number (one or more) of class 1 files. It is intended for broad, general alignments of any number of versions of any number of works. TAN-A-tok handle pairs (and only pairs) of class 1 files, and allows one to declare alignments with detailed specificity, certainty, and type.

Lexico-morphology files (also called part-of-speech files), TAN-LM, are used to encode the lexical headwords and morphological forms of individual words in class 1 files.

Common Elements

The class 2 formats have been designed with the goal of making the data as human readable as possible. And because that data invariably points to class 1 files, it has been assumed that those references should be as humanly intuitive as possible. In ordinary conversation, when we wish to point another person to a specific part of a text, we generally use locators that refer to pages, paragraphs, sentences, lines, words, letters, and so forth, and we include relative keywords, and sometimes portions of the text iteslf. For example, we prefer to say "See page 4, second paragraph, the last four words." Or "See page 4, second paragraph, first sentence, second occurence of 'pull'." We generally find it difficult if not impossible to construct and interpret "See characters 2496-2502, 2504-2509, and 2510-2512."

TAN adopts a system of referencing that imitates the form natural to humans. There are some unavoidable constraints. First, reference must be made to features that are encoded in the source XML file. So reference must be made to the reference system adopted by the source file, and not to altenative, overlapping reference systems. Second, because the sources are commited to space normalization and the flexibility therein entailed, it is impossible to point to space-sensitive features such as lines or character numbers. Therefore references to class 1 files depend upon four features that are relatively stable: work-versions, divisions, word tokens, and characters. *Work-versions* are taken to be equivalent to the *sources*, i.e., the class 1 files invoked by a class 2 file. *Divisions* are defined by the `<div>` structure of each class 1 transcription. *Tokens* are defined according to one or more tokenization rules. And *characters* are defined, when possible, by the codepoints that make up a particular word token.

This fourfold hierarchy—sources, divisions, tokens, and characters—are referred to with vocabulary that the editor of a class 2 file finds most useful. Sources can be given a nickname (through `@xml:id`), the names of division types and labels can be adjusted, word tokens can be referred to by number (whether from the beginning of a `<div>` or the end) or by content. Characters are always numbered.

This approach not only makes the syntax human readable, it also mitigates disruptions introduced by alterations in a source. For example, if a word in a source transcription is deleted, the token and character references might be disrupted, but references to the `<div>` or source remain valid.

The human-friendly reference system outlined above is facilitated by core components in the `<head>` and `<body>` of all class 2 files.

Class 2 Metadata (<head>)

All class 2 files have as their sources nothing other than class 1 files. Therefore each `<source>` must take only a single `<IRI>` that carries the IRI name of the source TAN-T(EI) file. There must be at least one `<location>`. Because the rights have already been declared in the source files, `<rights-source-only>` is not allowed. For more, see above, the section called "Sources and related files".

Within `<declarations>` each class 2 file takes the following children:

Table 7.1. Synopsis of **`<tokenization>`**, **`<suppress-div-types>`**, **`<implicit-div-type-refs>`**, **`<rename-div-types>`**, **`<rename-div-ns>`**, **`<rename>`**

name	req?	qty	parent	attributes	children	purpose
`<tokenization>`	varies	one	`<declarations>`	the section called "Edit Stamp"?, `@src`?, `@which`?	the section called "Digital Entity Metadata Pattern"?	indicate a tokenization rule to be applied to a source
`<suppress-div-types>`	no	one or more	`<declarations>`	the section called "Edit Stamp"?, `@div-type-ref`	data (xsd:string)	indicate what division types in what sources will be suppressed in references
`<implicit-div-type-refs>`	no	one	`<declarations>`	the section called "Edit Stamp"?, `@src`	----	specify sources whose div types should be inferred in references to that source
`<rename-div-types>`	no	one or more	`<declarations>`	the section called "Edit Stamp"?, `@src`	`<rename>`	Indicate, for a given source, div types whose `@xml:id` values should be provisionally renamed for the purpose of the class 2 file.
`<rename-div-ns>`	no	one or more	`<declarations>`	the section called "Edit Stamp"?, `@src`, `@div-`	`<rename>`	Indicate, for a given source and div type, `@n` values that should be provisionally renamed for the purpose of the class 2 file.

name	req?	qty	parent	attributes	children	purpose
				`type-ref`		
`<rename>`	yes	one or more	`<rename-div-types>` or `<rename-div-ns>`	`@old`, `@new`	----	Specify labels that should be renamed for the sake of the class 2 file.

Most class 2 files must be able to name or number word-tokens in a transcription. All class 2 files that require tokenization (i.e., TAN-A-tok and TAN-LM) must have under `<declarations>` one or more `<tokenization>`s that can be applied to every language in every source. Class 2 files that do not require tokenization (i.e., TAN-A-div) must include `<tokenization>` only if the file uses elements (e.g., `<split-leaf-div-at>`) that assume tokenization.

Each `<tokenization>` takes, in addition to the optional Edit Stamp, `@src` to specify which sources (suppressed in TAN-LM, which has only one source) and either `@which` or a the section called "Digital Entity Metadata Pattern" to a single TAN-R-tok file. The attribute `@which` takes either an id reference to a `<recommended-tokenization>` in the sources indicated by `@src` or a keyword to one of the core TAN tokenization files (`general-1`, `general-words-only-1`, and `precise-1`; see the section called "Recommended Tokenizations"). If neither the recommended nor core tokenization methods are suitable, you may point instead to another TAN-R-tok file through the the section called "Digital Entity Metadata Pattern". See the section called "Tokenization Patterns" for more on these files.

Caution

It is not predictable whether a regular expression engine will include or exclude combining characters in a search. You will need to worry about this only if you are working with texts that use combining characters. For more details see the section called "Regular Expressions" and Combining characters.

There may be some cases where a source has a div type that is extraneous to referencing the text. A single optional `<suppress-div-types>` may be used to specify division types that the aligner wishes to ignore. `<suppress-div-types>` takes, in addition to the optional Edit Stamp (the section called "Edit Stamp"), `@div-type-ref`, which takes one or more space-delimited id references to div types in the source file.

Caution

Any `<suppress-div-types>` invoked where omission of the div type does not preserve the Leaf Div Uniqueness Rule (see Flattened References, and the Leaf Div Uniqueness Rule) will result in an error upon validation.

A single `<implicit-div-type-refs>` is allowed. This otherwise empty element takes only `@src`, which lists the sources for which all div types will be suppressed in references to that source. This feature allows you to use briefer flattened references, but it does not preclude use of longer references (see the section called "`@ref`"). This feature is disallowed for sources where ignoring the types would result in duplicate flattened references and therefore ambiguity. For example, a source that has `question.13` followed by `answer.13` would make the simple flattened reference `13` impossible to interpret. The feature is also disallowed for sources that have an `@n` value that is empty. The only exception to this rule are cases where `<rename-div-ns>` is used to convert a numeration system to Arabic numerals; in those cases `n=""` will be interpreted to mean `n="0"`. For any `@src`, either all or no sources must be declared in `<implicit-div-type-refs>`. References to a mix of implicit and explicit div type sources are disallowed.

One or more `<rename-div-types>` may be invoked. `@src` specifies the applicable sources. The sole children of the element are one or more `<rename>`s, which through `@old` and `@new` specify the old label and the new one. One or more `<rename-div-ns>` may be used to swap the values of specific `@n`s for one or more div-types of one or more sources (defined through `@div-type-ref` and `@src`, respectively).

`<rename-div-types>` is useful for cases where an aligner wishes to use a different abbrevation for the same div type. This can be especially helpful in an alignment file, to avoid confusion in references to works that use different abbreviations for the same kind of division. Changing the div type does not affect the underlying definition. If you are writing a TAN-A-div file and you wish to equate div types defined differently (they do not have overlapping `<IRI>` values), you must use `<equate-div-types>`, discussed below (the section called "Division-Based Alignments (`<TAN-A-div>`)").

Similarly, `<rename-div-ns>` provides a convenient way to rename `@n` values. It may be useful in cases where you wish to use division labels that are more familiar, or that are identical to the vocabulary of another source invoked in the class 2 file. For example, if one source numbers the surahs of the Qur'an and another uses the traditional names, this element can be used to reconcile those differences, by assigning names to numbers, or vice versa. Or it can be used to adopt labels that would be easier to use in the alignment file. In addition, this feature can be used to convert Roman or alphabetic numerals to Arabic. In these cases, the content should be the special elements `<rename old="#i" new="#1"/>` (indicates a change from Roman numerals to Arabic) or `<rename old="#a" new="#1"/>` (change from letter numeration to Arabic).

`<rename-div-ns>` has potentially more impact than `<rename-div-types>`. Whereas the latter points to division types that are defined by the `<IRI>` value under `<div-type>`, and does not change that definition, the former points to and provisionally alters the value of `@n` in one or more `<div>`s. Because `<rename-div-ns>` effectively changes the actual value, it must preserve the Leaf Div Uniqueness Rule. Furthermore, in a TAN-A-div file, this kind of renaming changes the default behavior of the alignment. See the section called "Division-Based Alignments (`<TAN-A-div>`)".

Example 7.1. **`<filter>`**: **`<tokenization>`** through **`@which`**

```
...........
<declarations>
   ...........
   <tokenization src="A-fra" which="general-1"/>
   ...........
</declarations>
...........
```

Example 7.2. **`<declarations>`**: **`<tokenization>`** through IRI + name pattern

```
...........
<declarations>
   ...........
   <tokenization src="eng">
      <IRI>tag:textalign.net,2015:tokenization:eng.1</IRI>
      <name>Tokenization pattern for English, in Penn Treebank style</name>
      <location when-accessed="2015-03-17">../TAN-R-tok/tok.eng.1.xml</location
   </tokenization>
   ...........
</declarations>
```

```
...........
```

Example 7.3. **`<declarations>`**: **`<suppress-div-types>`**

```
...........
<declarations>
   ...........
   <suppress-div-types div-type-ref="stanza" src="ps-eng"/>
   ...........
</declarations>
...........
```

<caption>

In this example, Psalm 119 (LXX 118) has been subdivided by one TAN-T source into its traditional 22 stanzas (one for each letter in the Hebrew alphabet) before it has been subdivided into verses. The aligner chooses to suppress them, so that values of `@ref` can be more brief, e.g., `psalm.119:stanza.3:v.46` can be rendered simply as `psalm.119:v.46`).
</caption>

Example 7.4. **`<declarations>`**: **`<rename-div-types>`**

```
...........
<declarations>
   ...........
   <rename-div-types src="A-grc">
      <rename old="###" new="bk"/>
   </rename-div-types>
   ...........
</declarations>
...........
```

<caption>

The above `<rename-div-types>` changes the label for the division type book, from an abbreviation in the Greek alphabet to one in the Roman.
</caption>

Example 7.5. **`<declarations>`**: **`<rename-div-ns>`**

```
...........
<declarations>
   ...........
   <rename-div-ns src="A-grc" div-type-ref="bk">
      <rename old="###" new="Gen"/>
      <rename old="####" new="Ex"/>
      <rename old="###" new="Lev"/>
   </rename-div-ns>
   <rename-div-ns src="A-lat" div-type-ref="bk">
      <rename old="#i" new="#1"/>
   </rename-div-ns>
   ...........
</declarations>
...........
```

<caption>

This example illustrates both label changes and numeration system changes (from Roman numerals to Arabic).
</caption>

Class 2 Data Patterns (`<body>`)

Because all class 2 files have exclusively class 1 files as sources, some patterns recur throughout the `<body>` of class 2 files.

`<tok>`

Table 7.2. Synopsis of **`<tok>`**

name	req?	qty	parent	attributes	children	purpose
`<tok>`	yes	one or more	varies	the section called “Edit Stamp”?, `@cert`?*, `@src`, `@ref`, `@ord`?, `@val`?, `@chars`?	------	refer to a word token or word token fragment

* not true for TAN-A-div, which takes no certainty

`<tok>` takes, in addition to an optional the section called “Edit Stamp”, the following attributes:

`@cert` indicates the level of confidence an editor has in the statement made about the tokens referenced. This attribute is not allowed in the TAN-A-div format. See the section called “`@cert`”.

`@src` refers to one or more sources. This attribute does not exist in class 2 files with only one source (TAN-LM). In class 2 files with only two sources (TAN-A-tok), `@src` may take no more than one value. All others (TAN-A-div) allow multiple values.

`@ref` takes a sequence of references to sources. See the section called “`@ref`” for a full discussion.

One or more word tokens are referred to either through a numerical value (or range of numbers) for `@ord` or a string value for `@val` or a combination of the two (see the section called “`@ord` and `@val`” for the pattern).

`@chars` takes a sequence of numbers (constructed like `@ord`) that state an exact range of letters or characters (prefix, infixes, or suffix). This attribute is not allowed in TAN-A-div files.

Caution

`@chars` is not allowed for tokens that contain combining characters. See Combining characters.

There may be times when an editor needs a quick reminder as to how many words or characters are allowed. As a convenience, a question mark may be placed in `@ord` or `@chars`. Upon validation there will be both an error message for the malformed content (a question mark is not allowed) and a warning message indicating the maximal value allowed. The report does not take into account the presence of `@val`. That is, it returns the greatest value of `@ord` and `@chars` that is allowed.

`<tok>` therefore allows multiple values in each of four attributes, all distributed. That is, any assertion involving a `<tok>` applies to every set of characters in every word token in every `@ref` in every `@src`. For example, if `@src`, `@ref`, and `@ord` each have three values, then that `<tok>` names a sequence of twenty-seven tokens. (This allowance facilitates more terse and humanly readable `<body>`s.)

`@ref`

Recall that the transcriptions in TAN-T(EI) files are segmented and structured into a treelike hierarchy of nested `<div>`s, each one with a label (`@n`) and a division type (`@type`). The validation rules for class 1 files stipulate that leaf `<div>`s must have unique references. That is, for each leaf `<div>`, proceed from the highest ancestor `<div>` and move leafward, concatenating the values of `@type` and `@n` along the way. The result should be unique among all other leaf `<div>`s.

All class 2 files refer to parts of a class 1 source through a flattened reference (Flattened References, and the Leaf Div Uniqueness Rule). A flattened reference is the concatenated value of the `@type`s and `@n`s of all self and ancestral `<div>`s, joined by non-word characters as delimiters, and perhaps substituted with their new values if `<rename-div-types>` or `<rename-div-ns>` have been invoked (see the section called "Metadata (`<head>`)"). The delimiter to be used to separate `@n` and `@type` values and pairs of `@n` + `@type` values is completely arbitrary, but common punctuation is recommended. Hyphens and commas are allowed, but not recommended, since they are used specially to create ranges of references.

Take, for example, book 3 chapter 4 section 2 in a source transcription that uses the abbreviations `bk`, `c`, and `s` for the division types. When flattened, the reference becomes something like `bk.3:c.4:s.2`.

Punctuation in flattened references

The periods and colons in `bk.3:c.4:s.2` are entirely arbitrary. Other forms, such as `bk 3/c 4/s 2` `bk.3.c.4.s.2` `bk.3:c;4|s=2` are permissible and will be treated identically. But for the sake of those who must read your data, it is best to be clear and consistent and not to use misleading separators. Punctuation acts to delimit elements in a hierarchy, so the values being joined are in a hypotactic, not paratactic, relationship. You should avoid any punctuation that incorrectly suggests coordination, such as the ampersand or the comma. In these guidelines, a period is used to delimit `@n` and `@type` values, and a colon is used to chain `@n`-`@type` pairs.

It is important to avoid the use of the space as a delimiter in cases where unlabeled `<div>`s are used. For example, for an unnumbered preface, the preferred flattened reference would be `"preface."` and not `"preface "`.

Many sources will be structured to allow you to declare the div types to be implicit (see the section called "Class 2 Metadata (`<head>`)" [81]). In those cases either the longer form or a shorter one may be used: `3:4:2`.

In some cases, `@ref` may take a sequence of references, constructed by taking two or more flattened references and joining them with a hyphen `-` (for range) or a comma `,` (for union). In any range, the division referred to on the left should precede that on the right according to document order. Two flattened references joined by a hyphen signify an entire range of leaf `<div>`s. A comma indicates the union of references or ranges on either side. Flattened references must be spelled in full. For example, `bk.3:c.4:s.2 - bk.3:c.4:s.4 , bk.3:c.4:s.8` and `3:4:2 - 3:4:4 , 3:4:8` are correct ways to refer to sections 2, 3, 4, and 8 of the fourth chapter of book three.

In some cases, a range of references must be siblings (i.e., `bk.3 - bk.3:c.4:s.4` would be disallowed), as, for example, in elements for which `@distribute` (the section called "Data (`<body>`)" [101]) is `true`.

`@ord` and `@val`

To point to a token, one of three methods may be used.

1. *`@ord` alone*. Under this method, one or more digits, or the phrase `last` or `last-` plus a digit, joined by hyphens or commas indicate one or more token numbers. For example, `2, 4 - 6, last-2 - last` refers to the second, fourth, fifth, sixth, antepenult, penult, and final tokens in a sequence of word tokens.
2. *`@val` alone*. Under this method, a single token is picked by means of a string value equivalent to the token. For example, `@val = "bird"`, points to the first occurence of the token `bird`.
3. *`@ord` and `@val` together*. Under this method, specific occurences of a token are picked. For example, `@val="bird" @ord="2, 4"` picks the second and fourth occurences of the token `bird`.

If `@ord` points to an ordinal value of a word token or token value beyond the acceptable range for every `@src` and `@ref`, an error will be generated. This is true not just for absolute digits, but those combined with `last`. For example, the attribute combination `val="bird" ord="last-5"` will produce an error if the word token `bird` does not occur more than five times.

The numerical value to which the keyword `last` resolves depends upon the context of each source and ref.

There may be times when an editor needs a quick reminder as to how many words or characters are allowed. As a convenience, a question mark may be placed in `@ord` (the following is true also for `@chars`). Upon validation there will be both an error message for the malformed content (a question mark is not allowed) and a warning message indicating the maximal value allowed. The report does not take into account the presence of `@val`. That is, it returns the greatest value of `@ord` that is allowed.

`@cert`

`@cert` takes a decimal numeral from 0 to 1 or else `high` or `low`, which are to be interpreted as meaning "is probably" and "is probably not" respectively.

Assertions of certainty apply to a given statement absent inheritable attributes (see the section called "Inheritable attributes"), which establish assumptions made for a given range of data before confidence is expressed. Rather, `@cert` is to be interpreted to mean: "Assuming the abovementioned inheritable attributes to be true, I have `@cert` confidence about the following data:...."

Inheritable attributes

For class 2 formats, `<body>` and its descendants may take attributes whose values are defined in `<head>`. This special class of attributes are termed *inheritable attributes*. Each descendant of `<body>` takes the value of the attribute found in the closest ancestor, if not present in the element itself.

Each TAN format has been designed to limit inheritable attributes to the most critical concepts for the field of inquiry covered by that format. From the perspective of computability, there is no reason to restrict the number of inheritable attributes; from a human perspective there is, because each new inheritable attribute introduces a new parameter that has the potential to exponentially increase file complexity.

Every inheritable attribute is asserted without qualification. That is, if a inheritable attribute occurs alongside or under `@cert`, the latter does not apply to the former. Put another way, the combination of `@cert` and a inheritable attribute `@x` are to be interpreted to mean: ”Assuming `@x` to be true, I have `@cert` confidence about the following data:....”

TAN inheritable attributes are as follows:

Table 7.3. Inheritable attributes

Attribute name	A-tok	A-div	LM	refers to **`xml:id`** of	specifies
`@ed-who`	y	y	y	`<agent>`	Agent responsible for the data. See the section called “Agents”.
`@reuse-type`	y	–	–	`<reuse-type>`	The kind of textual reuse activity under consideration. See the section called “`<reuse-type>`”.
`@bitext-relation`	y	–	–	`<bitext-relation>`	The relationship between two source-bearing objects. See the section called “`<bitext-relation>`”.
`@lexicon`	–	–	y	`<lexicon>`	Thet lexicon that is being used. See the section called “`<lexicon>`”.
`@morphology`	–	–	y	`<morphology>`	The rules for morphology or parts of speech. See the section called “`<morphology>`”.

Alignments: Principles and Assumptions

At the heart of class 2 are alignment files. TAN distinguishes two types of alignment.

The first, aiming at breadth and extent, collects multiple versions of a work to present or study them in parallel. Examples from antiquity include the Rosetta Stone and Origen’s *Hexapla*. This macroscopic alignment helps a reader notice similarities and differences among comparable texts.

The second, aiming at depth and precision, isolates two versions of a work and explains how and why certain letters, words, or phrases in one version were omitted, retained, or changed in the other. This microscopic alignment characterizes the modus operandi of very precise, detailed commentaries on texts and their translations. One could also see the apparatus criticus of a critical edition as a kind of microscopic alignment.

Although the two types of alignment can be easily combined, in practice they tend to be separate. Similarly, TAN alignment has been distinguished in two formats, one, TAN-A-div, to facilitate the general alignment of any number of class 1 files; the other, TAN-A-tok, the detailed alignment of any two. For cases where both precision and breadth are required, multiple files of the latter should be used.

TAN aligments attest to acts of translating, paraphrasing, revising, quoting, summarizing, and so forth. All these are treated as types of text reuse, where one or more texts, usually called in translation studies the *source* (or *sources*), are transformed into a new text, customarily called the *target*. Text reuse has directionality and is asymmetrical. But many times we deal with texts where the original lines of direction are contested or unknown. In those cases, it is misleading to refer to either of the texts as a source or a target. Indeed, they may in fact derive from a common source, or be the result of multiple generations of copying and translating. In these guidelines, therefore, we avoid the term *target* altogether, and when we use the word *source*, we are referring only to one of the class 1 files upon which the alignment depends.

Consequently, the first `<source>` in an alignment file's `<head>` does not necessarily point to a version older than the others. But it will be processed first, and will therefore be the foundation or base against which subsequent sources will be aligned. It is usually a good idea to list as the first `<source>` the version that is most complete or most important to a given alignment.

Take for example a work such as the *Life of Saint Paul* (ca. early fourth century CE), which survives in both ancient Greek and Latin versions. The original language of composition is disputed. A TAN-A file that lists the Greek source first and the Latin second does not imply that the text was written in Greek. All it means is that anyone processing the TAN file should begin with the Greek before going to the Latin.

Division-Based Alignments (`<TAN-A-div>`)

The TAN format for macroscopic, division-based alignment, TAN-A-div, is dedicated to aligning any number of versions of any number of works on the basis of the `<div>`s in the sources. A TAN-A-div file is technically not an alignment per se. Rather, it is an alignment constructor. That is, a TAN-A-div file specifies parameters to be followed to generate a desired alignment. It is up to a processor to use that TAN-A-div file to create a byproduct (HTML, pdf, word processing document, etc.) that will be recognizable to most people as an alignment.

Transcribers, despite their best intentions, may fail to produce transcriptions of the same work that align. Perhaps the works or div types are not defined identically, or perhaps one version follows a reference system at odds with another. TAN-A-div has been designed to allow you to reconcile inconsistencies, and to explicitly declare alignments that a computer might not be able to detect automatically.

Every source invoked by a TAN-A-div will have alignments inferred wherever possible, and when not overridden explicitly. If the sources do not align exactly, or as one wishes, the format may be used to adjust and refine the alignment of the sources, even down to the word level.

Rootmost Element and Header

The rootmost element of a TAN division-based alignment file is `<TAN-A-div>`.

Under `<head>`, some special rules apply to TAN-A-div types.

One or more `<source>`s must be declared (the section called “Sources and related files”). It may seem strange to have an alignment file with only a single source, but this is allowed, to facilitate self-alignment (i.e., to indicate places where a source reuses itself). Every `<source>` must take an `@xml:id` for references to the source.

`<declarations>` takes zero or more of the four declarations common to class 2 files: `<tokenization>`, `<suppress-div-types>`, `<rename-div-types>`, `<rename-div-ns>`. See the section called “Common Elements”.

`<rename-div-ns>` can be used on occasion for simple points of reconciliation, and therefore alignment. For example, if two versions of the same work assign the same book two different `@n` values, e.g., "john" and "jn" for the New Testament gospel, then this element can be used to reconcile those differences. But it should not be used to reconcile more complex differences. And although it can be used to reconcile alternative numbering systems, that is a convenience, not a necessity of the alignment. Roman and alphabetic numbering systems are to be detected by preprocessors and to be treated identically to their Arabic numeral counterparts.

The assumptions of an aligner are opaque in the TAN-A-div format. A TAN-A-div file says, in essence, "Please align the following sources," but it does not say why the alignment is requested, and it does not make any statements about the relationship that holds between the various sources. Indeed, a TAN-A-div file could be used to align passages that have no apparent relationship (to what end might be unclear). If it is important to convey assumptions about how the text has been reused, the TAN-A-tok format should be used.

Example 7.6. TAN-A-div **<head>**

```
      ...........
   <head>
      <name>div-based alignment of multiple versions of Ring o Roses</name>
      <rights-excluding-sources rights-holder="park">
         <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
         <name>Creative Commons Attribution 4.0 International License</name>
         <desc>This data file is licensed under a Creative Commons Attribution 4.0
             License. The license is granted independent of rights and licenses as
             the source. </desc>
      </rights-excluding-sources>
      <source xml:id="eng-uk">
         <IRI>tag:parkj@textalign.net,2015:ring01</IRI>
         <name>Transcription of ring around the roses in English (UK)</name>
         <location when-accessed="2015-03-10">../TAN-T/ring-o-roses.eng.1881.xml</
      </source>
      <source xml:id="eng-us">
         <IRI>tag:parkj@textalign.net,2015:ring02</IRI>
         <name>Transcription of ring around the roses in English (US)</name>
         <location when-accessed="2014-08-13">../TAN-T/ring-o-roses.eng.1987.xml</
      </source>
      <source xml:id="ger">
         <IRI>tag:hans@beispiel.com,2014:ringel</IRI>
         <name>Transcription of an ancestor of Ring around the roses in German</na
         <location when-accessed="2014-08-13">http://beispiel.com/TAN-T/ringel.xml
         <location when-accessed="2014-08-13">../TAN-T/ring-o-roses.deu.1897.xml</
      </source>
      <source xml:id="eng-1790">
         <IRI>tag:parkj@textalign.net,2015:ring04</IRI>
         <name>TAN transcription of 1790 version of Ring around the Rosie reported
             1883</name>
         <location when-accessed="2014-10-28">../TAN-T/ring-o-roses.eng.1957.xml</
      </source>
      <declarations/>
      <agent xml:id="park" roles="creator">
         <IRI>tag:parkj@textalign.net,2015:self</IRI>
         <name xml:lang="eng">Jenny Park</name>
      </agent>
      <role xml:id="creator">
         <IRI>http://schema.org/creator</IRI>
         <name xml:lang="eng">creator</name>
      </role>
      <change when="2014-08-14" who="park">Started file</change>
   </head>
      ...........
```

Example 7.7. TAN-A-div **`<declarations>`**

```
   ...........
<declarations>
   <suppress-div-types src="Pss-heb" div-type-ref="stanza"/>
   <tokenization src="Pss-heb Pss-grc" which="general-words-only-1"/>
   <rename-div-types src="Pss-grc">
      <rename old="ch" new="ps"/>
   </rename-div-types>
</declarations>
   ...........
```

Data (<body>)

Table 7.4. Synopsis of TAN-A-div **`<body>`**

name	req?	qty	parent	attributes	children	purpose
`<body>`	yes	one	`<TAN-A-div>`	the section called "@in-progress"?, the section called "Edit Stamp"?	`<equate-works>`*, `<equate-div-types>`*, `<split-leaf-div-at>`*, `<realign>`*, `<align>`*	contains data

The TAN-A-div `<body>` may take the section called "@in-progress". It may take an edit stamp (see the section called "Edit Stamp"). It may take various elements, described below. But all these are optional.

A TAN-A-div may have an empty `<body>` because the format by default requests a greedy alignment. That is, it states, in effect, "Take the list of sources in the header. First group (align) them by work, then by `<div>`s according to flattened references."

A processor will create groups of works by looking for matches in the sources' `declarations/work/IRI` values. To those matches will be added any sources you claim should be treated as having equivalent works. Then within each work group, the processor will standardize the `@type` and `@n` values that make up each leaf `<div>`'s reference, first by using the sources' `declarations/div-type/IRI` values to create groups of division types and second by normalizing any nonstandard numeration systems. The resultant normalization is then used to create groups of leaf `<div>`s that share the same flattened reference numbers. (See Flattened References, and the Leaf Div Uniqueness Rule.)

If sources to be aligned share `<IRI>` values for `<work>`s and `<div-type>`s, and use a common labeling system (`@n`), then nothing needs to be declared in a TAN-A-div `<body>`. A TAN-conformant processor will construct an alignment wherever it can. `<body>` is used exclusively to reconcile unwanted differences between source TAN-T(EI) files, to realign versions of the same work, and to declare new alignments across works (quotations).

Table 7.5. Synopsis of **`<equate-works>`**, **`<equate-div-types>`**, **`<split-leaf-div-at>`**, **`<realign>`**, **`<align>`**, **`<div-type-ref>`**, **`<anchor-div-ref>`**, **`<div-ref>`**

name	req?	qty	parent	attributes	children	purpose
`<equate-works>`	no	one or more	`<body>`	`@src`, the section called "Edit Stamp"?	----	declares an ad hoc equivalence between sources whose `<work>`s are not defined as identical
`<equate-div-types>`	no	one or more	`<body>`	the section called "Edit Stamp"?	`<div-type-ref>`+	declares an ad hoc equivalence between div types that are in different sources and that are not defined as identical by those sources' `<div-type>`s.
`<split-leaf-div-at>`	no	one	`<body>`	the section called "Edit Stamp"?	the section called "`<tok>`"+	collects ad hoc splits in leaf `<div>`s
`<realign>`	no	one or more	`<body>`	the section called "Edit Stamp"?	`<anchor-div-ref>`?, `<div-ref>`+	eliminates or realigns the default alignment of divs in a source to other sources that share the same work
`<align>`	no	one or more	`<body>`	`@xml:id`?, `@strength`?, `@exclusive`?, `@distribute`, the section called "Edit Stamp"?	`<div-ref>`+	declares an ad hoc alignment of specific divs
`<div-type-ref>`	yes	two or more	`<equate-div-types>`	`@src`, `@div-type-ref`the section called "Edit Stamp"	----	references a div type
`<anchor-div-ref>`	yes	one	`<realign>`	`@src`, `@ref`, the section called "Edit Stamp", `@seg`?	----	references a div to which other `<div-ref>`s should be realigned
`<div-ref>`	yes	two or more	`<realign>`, `<align>`	`@src`, `@ref`, `@strength`?, the section called "Edit Stamp", `@seg`?	----	references a div

A TAN-A-div alignment permits an aligner to reconcile otherwise discordant sources in a five-step process. Each step is optional and sequence-specific. That is, each statement assumes declarations made in previous siblings have already been taken into account. Each of these steps allow an optional the section called "Edit Stamp" (therefore not mentioned in the description below).

Because the description of the theory behind these actions is abstract and difficult to understand, each of the steps is first provided examples, to make the theory more intelligible.

Step 1: correlate works

Example 7.8. `<equate-works>`: similar and related works defined differently

```
   ...........
<body>
   <equate-works src="lxx kjv nt"/>
</body>
   ...........
```

<caption>

The Bible's complex history has produced many different concepts and definitions of how that work should be defined. Take for example the Greek Septuagint (`lxx`), the New Testament (`nt`), and the King James Bible (`kjv`), each one justifiably defined as a distinct work. The `nt` is contained wholly by the `kjv`, and has no overlap with the `lxx` (ignoring for now `nt` quotations of the `lxx`). The `kjv` overlaps with only the majority of the `lxx` (books or parts of books commonly transmitted in the Septuagint translation never made it into the King James Bible). Since these three sources can be thought of as separate works, the creators of the TAN-T(EI) filse have each assigned a distinct `<IRI>` value to `<work>`. But for the purposes of this alignment, all three can be productively treated as a common work. The line above, `<equate-works>`, illustrates establishes an ad hoc equivalence among all three works. Any processor will make alignments wherever it can.
</caption>

Example 7.9. `<equate-works>`: different works

```
   ...........
<body>
   <equate-works src="hamlet lear othello"/>
</body>
   ...........
```

<caption>

This example illustrates an ad hoc alignment of Shakespeare's plays *Othello*, *Hamlet*, and *King Lear*. Any processor of this file will attempt to align the parts of the three plays that share the same reference number. This may be useful for an aligner who wishes to study the distribution of line length in the course of Shakespeare's plays by act, scene, and so forth.
</caption>

This first step allows you to declare an ad hoc equivalence between sources that do not already share a work `<IRI>` value. Each equivalence is made through an `<equate-works>`, which takes a mandatory `@src` that refers to one or more `<source>`s (by their ID). This element is otherwise empty.

As mentioned above, all TAN-A-div files assume greedy, transitive alignment. Therefore, to equate work X of source A with work Y of source B is to align all sources for works X and Y.

This declaration does not imply that the two works are, in reality, one and the same. It merely states that, for the purposes of this alignment, they should be treated as equivalent.

Step 2: correlate division types

Example 7.10. **`<equate-div-types>`**: common divisions defined differently

```
   ...........
<equate-div-types>
   <div-type-ref src="blue" div-type-ref="p"/>
   <div-type-ref src="red" div-type-ref="p"/>
</equate-div-types>
   ...........
```

<caption>

This example assumes two sources (`<blue>` and `<red>`) that both use `p` as a division type, but have not used the same IRI to define what `p` means. Whether the discrepancy is based upon a principled disagreement, an accident, or a typographical error, the above says that the two types of divisions are to be treated as equivalent.
</caption>

Example 7.11. **`<equate-div-types>`**: uncommon divisions defined differently

```
   ...........
<equate-div-types>
   <div-type-ref src="summa-eng" div-type-ref="contr"/>
   <div-type-ref src="summa-lat" div-type-ref="sedcontra"/>
</equate-div-types>
<equate-div-types>
   <div-type-ref src="summa-eng" div-type-ref="resp"/>
   <div-type-ref src="summa-lat" div-type-ref="respondeo"/>
</equate-div-types>
<equate-div-types>
   <div-type-ref src="summa-eng" div-type-ref="reply"/>
   <div-type-ref src="summa-lat" div-type-ref="adpraeterea"/>
</equate-div-types>
   ...........
```

<caption>

Many texts are traditionally segmented into divisions that are given names distinct to a particular discipline or culture. For example, the *Summa theologica* by Thomas Aquinas contains articles that allow five kinds of subdivision: question, objection, "on the contrary" (*sed contra*), "I respond that" (*respondeo*), and reply to objection. This example assumes an alignment of two independently transcribed versions of the *Summa theologica*. Both transcribers found IRIs suitable for the first two subdivisions (`http://dbpedia.org/resource/Question` and `http://dbpedia.org/resource/Objection_(argument)`) but they each had to mint their own IRI for the last three, and they do not match. This example says they should be treated as equivalent.
</caption>

This second step does for div types what the first step did for works. Across all sources, every `<div-type>` that shares an `<IRI>` value will be treated as equivalent. But if you wish to augment that automated alignment you may declare an ad hoc equivalence between `<div-type>`s that do not have any common `<IRI>` values. Each equivalence is made through an `<equate-div-types>`, which takes one or more `<div-type-ref>`s, each of which takes a mandatory `@src` and `@div-type-ref`, to point to one or more sources and division types. You must use either the `@xml:id` assigned by the source to that div type or its renamed value, if there is a corresponding `<rename-div-types>` under `<declarations>` (see the section called "Class 2 Metadata (`<head>`)").

As with `<equate-works>`, `<equate-div-types>` assume a greedy, transitive alignment. If div type X of source A is stated to be equivalent to type Y of source B, then all div types X and Y in all other sources are to treated as equivalent. This declaration does not imply that the two types of division are, in reality, one and the same. It merely states that, for the purposes of this alignment, they should be treated as equivalent.

Step 3: refine segmentation

Example 7.12. `<split-leaf-div-at>`: common use

```
      ...........
   <split-leaf-div-at>
      <tok src="eng-1790" ref="line 1" ord="1 | 2" val="a"/>
   </split-leaf-div-at>
      ...........
```

<caption>

This example assumes line 1 reads "Ring a ring a Rosie," and that the text has been tokenized based on spaces. This `<tok>` introduces two splits in the leaf `<div>`, starting with the first and second instances of "a" (the second and fourth word-tokens). The overall effect is to create three segments: "Ring" "a ring" "a Rosie". Later in the alignment file file, any reference to `line 1` will refer to all three segments. `@seg` would be used to point to one or more specific segments (see below).
</caption>

Example 7.13. `<split-leaf-div-at>`: advanced use

```
      ...........
   <split-leaf-div-at>
      <tok src="eng deu heb" ref="bk.Ex.ch.1.v.2 | bk.Ex.ch.1.v.1" ord="2 | 5 - 7
   </split-leaf-div-at>
      ...........
```

<caption>

In this example, the `<tok>` states that in each of the three different sources, each of the divisions corresponding to the two references should be split five times, at the second, fifth, sixth, seventh, and penultimate word-tokens. Note, there must be a valid word token for each ref in each source.
</caption>

Suppose you have two transcriptions where a phrase ending one leaf `<div>` in source A actually corresponds to the beginning phrase of the next leaf `<div>` in source B. Or suppose that you wish to break down a leaf `<div>` into smaller constituent parts, to facilitate more exact alignments (e.g., sentence-to-sentence, phrase-to-phrase, word-to-word). Before these refined alignments can occur, the leaf divs must first be segmented. A `<split-leaf-div-at>` may be used, containing one ore more `<tok>`s (see the section called "`<tok>`"), whose mandatory `@src` points to one or more specific sources and `@ref` to one or more references.

`@ref` must refer only to leaf `<div>`s. See the section called "`@ref`" and Flattened References, and the Leaf Div Uniqueness Rule for the pattern and rules.

To split a leaf `<div>`, you must be able to number or name word tokens, so a `<tokenize>` element must be included under `<declarations>` (see the section called "Tokenization Patterns"). A leaf `<div>` is split by declaring one or more word-tokens that should begin a new segment in each reference in each source. That word token is declared either through a numerical value (or range of numbers) for `@ord` or a string value for `@val` or both. See the section called "`@ord` and `@val`" for the pattern.

Any leaf `<div>` may be split as many times as one wishes.

Step 4: realign incorrectly aligned divisions and segments

Example 7.14. **`<realign>`**: unanchored 1

```
      ...........
   <realign>
      <div-ref src="tamil" ref="bk.1:ch.4" seg="2"/>
   </realign>
      ...........
```

<caption>

This example states that the second segment in book 1 chapter 4 of the Tamil version of a work should not be automatically aligned with any other version of that work that shares the same reference.

This technique may be useful for version that have interpolations that should be exempt from automatic alignment with other version of the same work.
</caption>

Example 7.15. **`<realign>`**: unanchored 2

```
      ...........
   <realign>
      <div-ref src="jones-1884" ref="topic.5:q.4 - topic.6:q.1 , topic.8"/>
      <div-ref src="denton-1822" ref="question.7 - question.9"/>
      <div-ref src="denton-1822" ref="discussion.4"/>
   </realign>
      ...........
```

<caption>

This example says that both the 1822 and 1884 versions of a work should be exempt from any automatic alignment with other versions of the same work, and that each of the questions from topic 5, question 4 through topic 6, question 1 in one source should be matched with three questions in the other. It also matches topic 8 in one source with discussion 4 in the other. In all these cases, not only will those divisions be realigned, but automatic alignment will be applied to their descendants. For example, if there is a `topic.8:para.1` in `jones-1884`, it will be automatically matched with `discussion.4:para.1` in `denton-1822` (provided that there is a common underlying IRI definition of `para` in both sources). Note that the last two `<div-ref>`s could have been combined: `<div-ref src="denton-1822" ref="question.7 - question.9 , discussion.4"/>`.
</caption>

Example 7.16. **`<realign>`**: unanchored 3

```
      ...........
```

```
<split-leaf-div-at>
   <tok src="poem-A" ref="line 46" ord="6"/>
   <tok src="poem-B" ref="line 47" ord="3"/>
</split-leaf-div-at>
<realign>
   <div-ref src="poem-A" ref="line 46" seg="2"/>
   <div-ref src="poem-B" ref="line 47" seg="1"/>
</realign>
<realign>
   <div-ref src="poem-A" ref="line 47" seg="1"/>
   <div-ref src="poem-B" ref="line 47" seg="2"/>
</realign>
   ...........
```

<caption>

This example realigns the parts of two versions of a poem, where one phrase that appears at the end of line 46 in one version begins line 47 of the other. The aligner (1) declares two splits, one for each version and (2) declares the alignment of the two segments.
</caption>

Example 7.17. **<realign>**: unanchored 4

```
   ...........
<realign>
   <div-ref src="A-1794" ref="para 7 - para 9"/>
   <div-ref src="A-1885" ref="para 8 - para 9 , para 7"/>
   <div-ref src="A-1922" ref="para 12 - para 14"/>
</realign>
   ...........
```

<caption>

This example assumes three versions of a text, where paragraphs seven through nine of one version correspond to eight, nine, and seven of the other and twelve through fourteen of the other.
</caption>

Example 7.18. **<realign>**: anchored

```
   ...........
<equate-works src="lxx kjv"/>
<realign>
   <anchor-div-ref src="kjv" ref="ps.9:v.1 - ps.10:v.18"/>
   <div-ref src="lxx" ref="ps.9:v.2 - ps.9:v.39"/>
</realign>
<realign>
   <anchor-div-ref src="kjv" ref="ps.11 - ps.113 | ps.117 - ps.146"/>
   <div-ref src="lxx" ref="ps.10 - ps.112 | ps.116 - ps.145"/>
</realign>
<realign>
   <anchor-div-ref src="kjv" ref="ps.114:v.1 - ps.115:v.18"/>
   <div-ref src="lxx" ref="ps.113:1 - ps.113:26"/>
</realign>
<realign>
   <anchor-div-ref src="kjv" ref="ps.116:v.1 - ps.116:v.19"/>
```

```
      <div-ref src="lxx" ref="ps.114:v.1 - ps.115:v.10"/>
   </realign>
   <realign>
      <anchor-div-ref src="kjv" ref="ps.147:v.1 - ps.147:v.20"/>
      <div-ref src="lxx" ref="ps.146:v.1 - ps.147:v.9"/>
   </realign>
      ...........
```

<caption>

This example takes reconciles the numbering of the Psalms in the Septuagint with that which is in most common use today. The anchored technique has been used because this comes from an example where multiple versions are used, so only one, labeled `kjv`, has been picked as the anchor. The component parts of the Septuagint version will then align with any other version that aligns with the anchor.
</caption>

After step 3, some of the divisions and segments of a work may not be properly aligned. Segments newly created by `<split-leaf-div-at>`s may need to be realigned. Or perhaps one of the sources uses a reference system that is out of step with the others.

Step 4, `<realign>`, is used to reconcile alignments within a given work. N.B., to realign is to assume that there is an error in automatic alignments, and that for two sources to be aligned they must, at the very least, be the same work. Therefore, `<realign>` applies only to one or more versions of a single work. If it is used to realign sources that do not share the same work, an error will result. (For cross-work alignment, see step 5, below.) There are two types of realignment: anchored and unanchored.

An *unanchored realigment* consists of one or more `<div-ref>`s, each of which takes `@src` and `@ref` to point to one or more sources and flattened references, and perhaps `@seg`, to point to one or more specific segments created by a preceding `<split-leaf-div-at>`. For any range of references included by `@ref` (values separated by a hyphen), the left and right sides must be siblings (i.e., `<div-ref src="a" ref="bk.1 - bk.2:ch.1"/>` is invalid). A range in `@ref` individually picks out each sibling from the first to the last (i.e., `bk.1 - bk.3` picks three references individually).

If `@seg` is used, every `@ref` must point to a leaf `<div>`. Under a given `<realign>`, each source's total number of references (determined by every `@seg` (if present) in every `@ref` in every `<div-ref>` that invokes that source) must be identical, or an error will result. Furthermore, there must be enough `<split-leaf-div-at>` declarations for each `@ref` in each `@src`, or an error will result.

The effect of an unanchored `<realign>` is (1) to sever each referenced `<div>` from any automatic alignment and (2) to realign each source's nth reference (div or segment) with the nth reference in every other other source, if any, invoked in the `<realign>`. If only one source is invoked in a `<realign>`, only the first effect takes place: the references are effectively severed from any automatic alignment with any other version for that source. For every `<div>` in a new realignment, descendants will be automatically aligned.

An *anchored realignment* differs from the unanchored form in that it begins with a single `<anchor-div-ref>` as the first child of `<realign>`. It is exactly like `<div-ref>`, except that the mandatory `@src` may point only to a single source. `<anchor-div-ref>` is then followed by one or more `<div-ref>`s. Once again, each source's total number of references must be identical, since every source's nth reference will be realigned to the nth reference in the anchor.

The effect of an anchored `<realign>` is to move each referenced `<div>`'s automatic alignment to a different place. This method is helpful in alignments of large numbers of sources, where a few do not follow the majority pattern and need to be synchronized.

For both anchored and unanchored alignments alike, the order of `<div-ref>`s is immaterial. Any processor of your data will handle the references according to source document order.

Step 5: add new alignments

Example 7.19. **`<align>`**: standard quotation

```
      ...........
   <align>
      <div-ref src="lecture-bra" ref="para.4:sent.2"/>
      <div-ref src="hamlet-jap" ref="act I:scene 1:line 1"/>
   </align>
      ...........
```

<caption>

This example presumes a written lecture, in multiple versions, that sometimes quotes from *Hamlet*, also in various versions. `<align>` states that in the source `lecture-bra`, paragraph 4 sentence 2, there is a quote from *Hamlet*, designated in the source `hamlet-jap` as act I, scene 1, line 1. Anything that aligns with the first reference will align with anything that aligns with the second.
</caption>

Example 7.20. **`<align>`**: standard quotation, **`@distribute = true`**

```
      ...........
   <align distribute="true">
      <div-ref src="article-A" ref="sect.4:blockquote.2:para.1 - sect.4:blockquote
      <div-ref src="quran-1" ref="sura.4:verse.3 - sura.4:verse.6"/>
   </align>
      ...........
```

<caption>

This example assumes an article that quotes from the Qur'an, sura 3 (The Women), 3-6. If `@distribute` were `false` or missing, the references in `article-A` would be taken as aligning en bloc with the references in `quran-1`. But because `@distribute` is `true`, the effect is to create four separate, more specific, alignments.
</caption>

Example 7.21. **`<align>`**: standard quotation, **`@exclusive = true`**

```
      ...........
   <align exclusive="true">
      <div-ref src="S2" ref="century VI chapter 88"/>
      <div-ref src="lxx" ref="bk Judg.ch 5.v 20"/>
   </align>
      ...........
```

<caption>

This example illustrates a situation where one version of a work quotes from Judges 5:20 (Septuagint) but the other does not. Because `@exclusive` is set to `true`, no other versions will be automatically included in the alignment.
</caption>

Example 7.22. **`<align>`**: one work self-alignment, three options

```
   ...........
<align xml:id="lords-prayer-1">
   <div-ref src="nt-grc" ref="bk.Matt:ch.6:v.9 - bk.Matt:ch.6:v.13" />
</align>
<align xml:id="lords-prayer-2">
   <div-ref src="nt-grc" ref="bk.Luke:ch.11:v.2 - bk.Luke:ch.11:v.4" />
</align>
<align alignments="lords-prayer-1 lords-prayer-2"/>
   ...........

   ...........
<split-leaf-div-at>
   <tok src="nt-grc" ref="bk.Matt:ch.6:v.9 , bk.Matt:ch.6:v.10" ord="5"/>
   <tok src="nt-grc" ref="bk.Luke:ch.11:v.2" ord="7, last-3"/>
   <tok src="nt-grc" ref="bk.Luke:ch.11:v.4" ord="last" val="###"/>
</split-leaf-div-at>
<align xml:id="lords-prayer-1">
   <div-ref src="nt-grc" ref="bk.Matt:ch.6:v.9:seg.2"/>
   <div-ref src="nt-grc" ref="bk.Matt:ch.6:v.10:seg.1"/>
   <div-ref src="nt-grc" ref="bk.Matt:ch.6:v.11"/>
   <div-ref src="nt-grc" ref="bk.Matt:ch.6:v.12"/>
   <div-ref src="nt-grc" ref="bk.Matt:ch.6:v.13"/>
</align>
<align xml:id="lords-prayer-2">
   <div-ref src="nt-grc" ref="bk.Luke:ch.11:v.2:seg.2"/>
   <div-ref src="nt-grc" ref="bk.Luke:ch.11:v.2:seg.3"/>
   <div-ref src="nt-grc" ref="bk.Luke:ch.11:v.3"/>
   <div-ref src="nt-grc" ref="bk.Luke:ch.11:v.4:seg.1"/>
   <div-ref src="nt-grc" ref="bk.Luke:ch.11:v.4:seg.2"/>
</align>
<align alignments="lords-prayer-1 lords-prayer-2" distribute="true" exclusive="
   ...........

   ...........
<split-leaf-div-at>
   <tok src="nt-grc" ref="bk.Matt:ch.6:v.9 , bk.Matt:ch.6:v.10" ord="5"/>
   <tok src="nt-grc" ref="bk.Luke:ch.11:v.2" ord="7, last-3"/>
   <tok src="nt-grc" ref="bk.Luke:ch.11:v.4" ord="last" val="###"/>
</split-leaf-div-at>
<align xml:id="lords-prayer-1">
   <div-ref src="nt-grc"
      ref="bk.Matt:ch.6:v.9:seg.2 , bk.Matt:ch.6:v.10:seg.1 , bk.Matt:ch.6:v.11
   />
</align>
<align xml:id="lords-prayer-2">
   <div-ref src="nt-grc"
      ref="bk.Luke:ch.11:v.2:seg.2 , bk.Luke:ch.11:v.2:seg.3 , bk.Luke:ch.11:v.
      bk.Luke:ch.11:v.4:seg.1 , bk.Luke:ch.11:v.4:seg.2"
   />
</align>
<align alignments="lords-prayer-1 lords-prayer-2" distribute="true" exclusive="
   ...........
```

<caption>

This example shows three ways of self-aligning the the two versions of the Lord's Prayer in a single work, the New Testament.

The first method shows a rough alignment, connecting the five verses of Matthew with the three of Luke. This alignment provides only a rough alignment, but it will be applicable to any other version of the New Testament cited in the TAN-A-div file as a source.

The second and third alignments are equivalent ways of doing a more refined alignment. The only difference is that the second places each of the five parts of each version in its own `<div-ref>` whereas the third collapses each into a single `<div-ref>`. Legibility should be the key to deciding which method should be preferred.

The refined alignments begin by declaring two splits in the verses in Matthew (both verses 9 and 10 happen to have natural breaks at the fifth word token, excluding punctuation). The Luke version has three splits. Notice that the aligner has chosen not to mention Matthew 6.10b because there is no corresponding phrase in Luke. (Another aligner may have approached this differently.)

The refined alignment offers precision, but at the expense of broad applicability. Any other version of the New Testament included in the TAN-A-div file that has not been properly segmented would be misaligned. Therefore `@exclusive` is declared to be `true`, to exempt the alignment from being transitively applied to any other New Testament source.
</caption>

At this point, each work's versions should be aligned with each other. You are now in a position to declare new alignments to supplement the ones that will be automatically generated from steps 1 through 4.

The fifth step, one or more `<align>`s, is used normally when you wish to indicate where one work quotes another, or a single work repeats texts that merit alignment. Unlike `<realign>`, `<align>` does not negate any other alignments, and it allows sources from multiple works to be mentioned. Furthermore, whereas multiple values of `@ref` govered by a `<realign>` were taken individually, under `<align>`, any `@ref` will usually be treated as grouped by work, but it may also be taken individually (see the section called "Data (`<body>`)" [101]).

In addition, `<align>` takes several special attributes, all optional.

The first is `@strength`. As mentioned above, TAN-A-div files do not furnish a mechanism to indicate the reason why the sources are being aligned (e.g., quotation, translation), and so it does not admit any doubt. If you are aligning a text you are doing it with certitude. You are not committing yourself to any assertions in your alignment. Yet it is understood that some alignments are stronger than others. Verbatim quotations, for example, should be differentiated from indirect references. For these purposes, `@strength` may be used. It takes a rational decimal number from 0 to 1 stipulating how strong the alignment should be. The values are completely arbitrary, and strictly speaking no meaning is to be assigned to this attribute. In practice, however, this value will be a guage of how close the alignment is. If you need to specify exactly what concept of text reuse attaches to an alignment, you should use TAN-A-tok.

The second special attribute is `@exclusive`, which indicates through a boolean value whether an alignment is to be applied to other implicitly aligned work versions or not. If set to `true`, then the alignment is excluded from transitive alignments. Take, for example, work A and work B, each with three sources. If an `<align>` creates an alignment between passages in A1 and B1, then that alignment applies to A1-3 and B1-3 if `@exclusive` is `false` and it does not if it is `true`. The default value is `false`, because most of the time, when you say that one work quotes from another, you mean that statement to be true for all versions of the work. But on occasion, one versions of a work might have a quotation that is lacking in another. In these cases it is important to set `@exclusive` to `true`.

The third attribute is `@distribute`, which takes a boolean value. If `true`, then `@ref` functions identically to the way it does in step 4, under `<realign>`: each reference made by `@ref` or `@seg` will be treated individually. But if `false`, then the values of `@ref` or `@seg` will be treated as a group, one group per work. The default value is `false` because in most situations a single `<align>` will be dealing with only one quotation, and the intent is not to distribute each reference, but to group and then align them. But if you are aligning a series of quotations of one work in another, you may find that `@distribute` set to be `true` allows you collapse multiple `<align>`s into a single one. Keep in mind, though, that the total number of references per work must be identical, or the file will be invalid. Furthermore, if `@distribute` is true, then the beginning and ending references for any range in `@ref` (e.g., `ref.A - ref.C`) must be siblings, and will be treated as referring to the start of the range, the end of the range, the intermediate siblings (e.g., `ref.A - ref.C` will be treated as referring to three items: `ref.A`, `ref.B`, and `ref.C`).

A fourth attribute is an optional `@xml:id`, which allows you to name an alignment cluster. This is used principally by other `<align>`s that take the fifth optional attribute, `@alignments`, which points to two or more other `<align>`s by means of their `@xml:id` values. This is the only way you can create new alignments within a single work (self-alignment). For every appearance of a passage in a given work use a single `<align>`, give it an `@xml:id`, then create one more `<align>` that points through `@alignments` to all the relevant `<align>`s. But `@alignments` need not be used for self-alignment. It may point to any `<align>` that takes an `@xml:id`, no matter its content. But to avoid fatally recursive loops created by reflexive alignments, `@xml:id` and `@alignments` are mutually exclusive.

Every `<align>` that has `@alignments` is empty. Every other `<align>` takes one or more `<div-ref>`s, which indicates through mandatory `@src` and `@ref`, and optional `@seg`, one or more references that should be aligned (see the section called "Class 2 Data Patterns (`<body>`)").

The constraints on `<div-ref>` are identical to those in step 4 (see the section called "Data (`<body>`)" [97]) with one caveat. Any `<div-ref>` that is a child of an `<align>` may take the optional `@strength`. (If not present, the value is assumed to be `1`.) This feature is helpful when you wish to indicate in a quotation that one work does not align as strongly with the rest of the group as do the others. Note, in these cases the value of `@strength` is to be applied to the work, not just to the source. If you wish to differentiate between how strongly different versions of a work align with another work, you should create a separate `<align>` for each relevant parallel, with `@exclusive` set to `true`.

Overall, `<align>` can be thought of as a group of clustered references, one cluster per work. Those clusters may have constitutent references that are very small (e.g., a segment that is a single word) or quite large, and with descendants of its own (e.g., an entire book). A cluster could be a single reference or quite a large number of them. Therefore, clusters that have been grouped by `<align>` may be quite heterogenous in number, length, and hierarchy. And that is how it should be. A novel may in a single sentence synthesize from another work multiple passages of much greater length. The specifics of how the constituent parts of one cluster align with the constituent parts of another cannot be predicted or dictated. All an `<align>` does is say, in effect, "This group of passages from work A are to be aligned with this group of passages from work B." It is up to the processor of a TAN-A-div file to determine if the alignment should be taken *en bloc*, or if it should be distributed into individual parts, and if so, upon what basis. If you wish to control how that distribution occurs in a given `<align>`, you must break it up into separate individual `<align>`s.

As in step 4, the order of `<div-ref>`s in an `<align>` is immaterial. Any processor of your data will handle references according to the document order of the sources.

The five steps involved in creating the `<body>` for a TAN-A-div file are, in general, rather simple. If some of the description of this section seems unnecessarily detailed and complex, consult examples of TAN-A-div files.

Token-Based Alignments (`<TAN-A-tok>`)

TAN-A-tok files provide a microscopic view of how two texts relate to each other. The format is intended to allow you to specify exactly where, how, and why two texts align, and to do so on the most granular level possible. TAN-A-tok files also allow you to express levels of confidence or alternative opinions.

Creators and editors of TAN-A-tok files should be able to read the languages of their sources and to explain as precisely as possible the relationship between the two sources. You should be prepared to think about and specify types of textual reuse. TAN-A-tok files tend to be more demanding to create and edit than TAN-A-div files are because they support work that is more detailed, and therefore more time-consuming, than simple en masse alignment of sources.

Because of the detailed nature of the inquiry, token alignment is restricted to two texts, referred to jointly as a *bitext*. Each half of the bitext must be a TAN-T(EI) file. It is assumed that those two sources share some special relationship, direct or indirect, and are the results of certain types of textual reuse, sometimes mixed: translation, paraphrase, commentary, and so forth. Some of these bitexts, such as literal translations, may line up quite nicely word for word. Others, such as paraphrases, may line up sporadically, vaguely, or ambiguously. Some bitexts produce few word-for-word correspondences. Therefore alignment of a bitext is frequently not easy, and requires you as a TAN-A editor to declare the assumptions you have made in two key areas: the relationship that holds from one source to the other and the types of reuse that was involved in turning one version into the other (or a common ancestor into both).

Relationship of sources. Some bitexts will be directly related. Others will be removed from each other, or a common original, by several generations. And that history may itself be uncertain or vague. You may find it difficult if not impossible to tell whether a striking difference between the two source texts is to be credited/blamed on a translator or on an intermediary (such as an editor or redactor). What you know and understand about the relationship will affect how you create token alignments. Therefore, you must declare what you believe to be the physical relationship or history that connects the two sources.

Types of reuse. In creating a TAN-A file, you should decide the types of text reuse you wish to declare. Such a declaration gives users of your data a sense of the paradigm you bring to your analysis. You may wish to keep your categories nondescript and somewhat vague, using terms such as *translation*, *paraphrase*, *quotation*, and so forth without offering a specific definition. On the other hand, you may be specific and detailed, for example adopting field-specific categories such as *obligatory explicitation*, *optional explicitation*, *pragmatic explicitation*, or *translation-inherent explicitation*. You may also wish to declare secondary types of reuse, such as *scribal omission* or *dittography*, to declare secondary types of reuse that may have intervened. You must declare at least one type of reuse.

Rootmost Element and Header

The rootmost element of a token-based alignment file is `<TAN-A-tok>`.

The TAN-A-tok header builds upon the core and class 2 headers (see the section called “Metadata (`<head>`)” and the section called “Class 2 Metadata (`<head>`)”).

TAN-A-tok files take exactly two `<source>`s. The sequence is arbitrary, determining only the order in which most processors will handle the data. Each `<source>` must take an `@xml:id`.

`<declarations>` takes, in addition to all the elements allowed in class 2 files (see the section called "Class 2 Metadata (`<head>`)"), two elements unique to TAN-A-tok: `<bitext-relation>` and `<reuse-type>`. The first of these describes genealogical descent of the two versions, and is concerned with directionality. The second avoids these categories altogether and focuses on the qualitative aspect of the bitext relationship.

`<bitext-relation>`

Table 7.6. Synopsis of `<bitext-relation>`

name	req?	qty	parent	attributes	children	purpose
`<bitext-relation>`	yes	one or more	`<declarations>`	the section called "Edit Stamp"?, `@xml:id`	the section called "IRI + name Pattern"	names a relation that holds between two textual sources

`<bitext-relation>`s may take an the section called "Edit Stamp" and must take an `@xml:id`. `<bitext-relation>` otherwise consists of an the section called "IRI + name Pattern" that names a type of relationship that holds between any two texual objects. This relationship pertains to the

In most cases, there will be only a single `<bitext-relation>`. Multiple values are permitted for situations where a bitext has a complex history, for example, a textual object that was created over time, and in different phases.

There is, at present, no known stable, independent IRI vocabulary to describe bitext relations. Therefore a handful of TAN-minted iris.xml may be of assistance. It is hoped that use of this vocabulary will contribute to more formalized vocabularies in stemmatics.

Example 7.23. `<bitext-relation>`: unclear

```
   ...........
<bitext-relation xml:id="unclear">
   <IRI>tag:textalign.net,2015:bitext-relation:unclear</IRI>
   <name>unclear</name>
   <desc>The two versions bear some relationship, but what exactly is
       unclear.</desc>
</bitext-relation>
   ...........
```

Example 7.24. `<bitext-relation>`: direct descent

```
   ...........
<bitext-relation xml:id="b-x-a">
   <IRI>tag:textalign.net,2015:bitext-relation:b/x/a</IRI>
   <name>direct relationship, A descends from B, one mediary.</name>
   <desc>The first source comes from the second source, via a single intermedia
</bitext-relation>
   ...........
```

Example 7.25. `<bitext-relation>`: indirect descent

```
   ...........
<bitext-relation xml:id="b-x-a">
```

```
      <IRI>tag:textalign.net,2015:bitext-relation:/a,/x*/b</IRI>
      <name>parent of A is ancestor of B.</name>
      <desc>The first source comes directly from the direct ancestor of the second
         one or more intermediaries).</desc>
   </bitext-relation>
      ...........
```

<reuse-type>

Table 7.7. Synopsis of **<reuse-type>**

name	req?	qty	parent	attributes	children	purpose
`<reuse-type>`	yes	one or more	`<declarations>`	the section called “Edit Stamp”?, `@xml:id`	the section called “IRI + name Pattern”	names a type of text reuse

One or more `<reuse-type>`s, which take the the section called “IRI + name Pattern”, declare the types of text reuse that are relevant to the bitext. Each `<reuse-type>` must include `@xml:id`.

It is up to you to determine how specific or vague you wish to define your reuse types. If you prefer categories that are common to translation studies or textual criticism, you should probably employ terminology others are using, if it can be found. But if you find the terms to be poorly defined or difficult to locate, and you wish to define your own, you may coin your own through a UUID or a tag URN.

There is, at present, no known stable, independent IRI vocabulary to describe types of textual reuse. See the supplementary document [iris.xml] for a list of TAN-minted IRIs .

Example 7.26. **<reuse-type>**: quotation

```
      ...........
   <reuse-type xml:id="quotation">
      <IRI>tag:textalign.net,2015:reuse-type:quotation:general</IRI>
      <name>quotation, general</name>
      <desc>One version quotes from the other</desc>
   </reuse-type>
      ...........
```

Example 7.27. **<reuse-type>**: translation in general

```
      ...........
   <reuse-type xml:id="trans-gen">
      <IRI>tag:textalign.net,2015:reuse-type:translation:general</IRI>
      <name>general translation</name>
      <desc>One version is a translation of the other.</desc>
   </reuse-type>
      ...........
```

Example 7.28. **<reuse-type>**: paraphrastic translation

```
      ...........
```

```
<reuse-type xml:id="trans-loose">
   <IRI>tag:textalign.net,2015:reuse-type:translation:paraphrastic</IRI>
   <name>paraphrastic translation</name>
   <desc>One version is a loose translation of the other.</desc>
</reuse-type>
   ...........
```

Data (<body>)

Table 7.8. Synopsis of TAN-A-tok **`<body>`**, **`<align>`**

name	req?	qty	parent	attributes	children	purpose
`<body>`	yes	one	`<TAN-A-tok>`	the section called "@in-progress"?, the section called "Edit Stamp"?, `@bitext-relation`, `@reuse-type`	`<align>`+	contains data
`<align>`	yes	one or more	`<body>`	`@xml:id`?, the section called "Edit Stamp"?, `@cert`?, `@bitext-relation`?, `@reuse-type`?	the section called "`<tok>`"+	aligns one or more tokens

The `<body>` of a TAN-A-tok file takes, in addition to the customary optional attributes found in other TAN files (see the section called "@in-progress" and the section called "Edit Stamp"), required `@bitext-relation` and `@reuse-type`, which take one or more id references from `<bitext-relation>` and `<reuse-type>`, defining the defaults of the alignment. `<body>` has only one type of child: one or more `<align>`s.

`<align>` may take an the section called "Edit Stamp" as well as `@cert` to indicate certainty or confidence (see the section called "`@cert`"). An optional `@xml:id` allows you to refer to a specific alignment (see possible uses below). An optional `@reuse-type` and `@bitext-relation` may be used to override the default values in `<body>`.

`<align>` has only one type of child: one or more `<tok>`s (see the section called "`<tok>`"). These collectively form a *token cluster*. Each token cluster is valid independent of the construction of any other token cluster. Token clusters overlap in part, in whole, or not at all. This reflects the nature of translations, in which words can easily fall in one-to-one, one-to-many, many-to-one, and many-to-many relationships. Or there may be differences of opinion, and you may wish to register different opinions. If you supply an `<align>` with an `@xml:id` then you effectively name your token cluster, and permit references to it, and therefore discussions about it.

Unlike TAN-A-div files, under which a processor will assume automatic, transitive, greedy alignment, TAN-A-tok files specify the opposite. Nothing should be inferred about two texts outside of explicitly stated alignments. If a TAN-A-tok file does not mention word tokens that are in a source, it does not mean that they were, for example, left out of a translation. All it means is that the creators and editors of the TAN-A-tok file have nothing to say about the tokens.

If you wish to declare that one or more words in one source were left out of a translation or inserted into one, you must do so through a *half-null alignment*, i.e., a token cluster that has tokens from one source but not the other. That is, a half-null alignment implies—to drawn from the terminology of translation studies—implicitation or explicitation for a given type of text reuse (defined by `@reuse-type`).

Because nothing can be implied in a TAN-A-tok file, the status of `@in-progress` (default = true) is very important, since it is the only way to signal that you have said all you intend to say about the bitext. (Remember, `in-progress = "false"` does not mean that the file is free of errors, only that the entire scope of work has been finished.)

A fully aligned bitext may result in a TAN-A-tok file with a very long `<body>` (in contrast to a typical TAN-A-div file). That does not mean, however, that everything in a transcription *must* be encoded or described. A TAN-A-tok file does not commit you to saying everything possible about the bitext. You might choose to encode only a few token clusters.

`@bitext-relation` and `@reuse-type` are inheritable attributes (see the section called "Inheritable attributes"). That is, they declare the assumptions made for a given range of data. Any `@cert` does not express certainty about these values. Rather, `@cert` is to be interpreted to mean: "Assuming the abovementioned `@bitext-relation` and `@reuse-type` to be true, I have `@cert` confidence about the following assertions:...."

If there are multiple IDs in `@reuse-type` or `@bitext-relation`, the union, not the intersection, of those values is to be understood. For example, `reuse-type="trans para"` might be used to indicate that the token cluster results from translation, paraphrase, or both.

Example 7.29. TAN-A-tok `<body>`

```
  ...........
<body bitext-relation="unclear" reuse-type="adaptation" in-progress="false">
    <align>
        <tok src="ring1881" ref="line 1" ord="1"/>
        <tok src="ring1987" ref="l 1" ord="1"/>
    </align>
    <align>
        <tok src="ring1881" ref="line 2" val="pocket"/>
        <tok src="ring1987" ref="l 2" val="pocket"/>
    </align>
    <align>
        <tok src="ring1881" ref="line 3" ord="1" cert="high"/>
        <tok src="ring1881" ref="line 3" ord="2"/>
        <tok src="ring1987" ref="l 3" ord="1"/>
    </align>
</body>
  ...........
```

Lexico-Morphology

TAN-LM files are used to associate words or word fragments with lexemes and morphological categories. They are intended primarily to facilitate research that depends upon alignments, but they can interesting in their own right, whether or not there are other versions or alignments.

These files are indelibly reliant upon the grammatical rules declared for a given language, so these instructions should be read in close conjunction with the section devoted to the TAN-R-mor format (see the section called "Morphological Concepts and Patterns (TAN-R-mor)").

Principles and Assumptions

TAN-LM files are assumed to be applicable to texts in languages whose vocabulary lends itself to grammatical and lexicographical analysis. The two areas are interrelated but can be treated independently. If you wish, your TAN-LM file may contain only lexemes or only morphological analyses.

As an editor of a TAN-LM file you should understand the vocabulary and grammar of the languages you have picked. You should have a good sense of the rules established by the lexical and grammatical authorities you have chosen to follow.

Although you must assume the point of view of a particular grammar and lexicon, you need not define those authorities, nor hold to a single one. You at liberty to choose the morphological scheme that best suits your opinions or purposes, as defined in one or more TAN-R-mor files. In addition, you may bring to lexical analysis your own expertise and supply lexical headwords unattested in printed authorities.

Although TAN-LM files are simple, they can be laborious to read and write, more than other types of TAN files. It is expected that an editor of a TAN-LM file will find one or more tools to help create and edit the data.

Rootmost Element and Header

The rootmost element of a lexico-morphological file is TAN-LM.

A mandatory `<source>` element points to the one and only TAN-T(EI) file that is the object of analysis.

`<declarations>` takes the elements common to class 2 files (see the section called "Class 2 Metadata (`<head>`)". It takes two other elements unique to TAN-LM: `<lexicon>` and `<morphology>`.

`<lexicon>`

Table 7.9. Synopsis of **`<lexicon>`**

name	req?	qty	parent	attributes	children	purpose
`<lexicon>`	no	one or more	`<declarations>`	the section called "Edit Stamp"?, `@xml:id`	the section called "IRI + name Pattern", the section called "Digital Entity Metadata Pattern"?	names a lexicon that has been adopted

Each `<lexicon>` may take an the section called "Edit Stamp" and must take an `@xml:id`. The element otherwise consists of an the section called "IRI + name Pattern" that names a lexicon that has been adopted in the lexical analysis.

You may include any number of lexica, which may be listed in any order you wish. This will be important for source texts or languages where one lexicon is not sufficient (true especially for medieval vernacular texts or for poorly resourced modern languages). `<IRI>` will most often point to URL names (e.g., a Library of Congress Control Number for an authority in print) or some other unique identifier for the authority used. If you or another editor are a lexical authority, no separate `<lexicon>` entry need be given. Credit will be made to the appropriate `<agent>`.

There is, at present, no standard exchange format for lexica and dictionaries. So even if a digital form of a dictionary is identified through the the section called "Digital Entity Metadata Pattern", no validation tests will be performed.

Example 7.30. **<lexicon>**

```
   ...........
<lexicon xml:id="LSJ">
   <for-lang>grc</for-lang>
   <IRI>http://lccn.loc.gov/95032369</IRI>
   <name xml:lang="eng">Liddell-Scott-Jones, 9th ed. plus rev. supplement</name
</lexicon>
   ...........
```

<morphology>

Table 7.10. Synopsis of **<morphology>**

name	req?	qty	parent	attributes	children	purpose
`<morphology>`	yes	one or more	`<declarations>`	the section called "Edit Stamp"?, `@xml:id`	the section called "IRI + name Pattern", the section called "Digital Entity Metadata Pattern"	names a TAN-R-mor file that has been adopted

Each `<morphology>` may take an the section called "Edit Stamp" and must take an `@xml:id`. The element otherwise consists of a the section called "Digital Entity Metadata Pattern" that names and points to a TAN-R-mor file.

Morphologies may be listed in any order you wish.

In most cases, there will be only one `<morphology>` declaration. Multiple values are permitted for texts with multiple languages, or for cases where you wish to register alternative analyses.

Example 7.31. **<morphology>**

```
   ...........
<morphology xml:id="Perseus">
   <for-lang>grc</for-lang>
   <IRI>tag:jenny.park@example.com,2014:tan-m-rules:grc:perseus</IRI>
   <name xml:lang="eng">Perseus Greek morphology</name>
   <location when-accessed="2015-03-02">../TAN-R-mor/grc-1.xml</location>
</morphology>
   ...........
```

Data (<body>)

Table 7.11. Synopsis of TAN-LM **<body>**, **<ana>**, **<joined>**, **<lm>**, **<l>**, **<m>**

name	req?	parent	attributes	children	purpose
`<body>`	yes	`<TAN-LM>`	the section called "@in-progress"?, the section called "Edit Stamp"?, `@lexicon`, `@morphology`	`<ana>`+	contains data

name	req?	parent	attributes	children	purpose
`<ana>`	yes	`<body>`	the section called "Edit Stamp"?, `@cert`?, `@lexicon`?, `@morphology`?	(`<tok>`|`<joined>`)+, `<lm>`*	aligns one or more tokens
`<joined>`	no	`<ana>`	`@xml:id`?	`<tok>`, `<tok>`+	joins two or more strings that should be joined and treated as a single linguistic unit
`<lm>`	yes	`<ana>`	the section called "Edit Stamp"?, `@cert`?	`<l>`*, `<m>`*	assert a lexico-morphological combination
`<l>`	no	`<lm>`	the section called "Edit Stamp"?, `@cert`?, `@lexicon`?, `@ord`?	data (xsd:string)	assert a lexeme
`<m>`	no	`<lm>`	the section called "Edit Stamp"?, `@cert`?, `@morphology`?	data (xsd:string)	assert a morphological pattern

The `<body>` of a TAN-LM file takes, in addition to the customary optional attributes found in other TAN files (see the section called "@in-progress" and the section called "Edit Stamp"), two required attributes. `@lexicon` points to one or more lexica via `<lexicon>` ids; `@morphology` takes only one morphology id. These values define the default lexica and grammar of the analysis.

`<body>` has only one type of child: one or more `<ana>`s.

`<ana>` (short for analysis) takes an optional the section called "Edit Stamp" as well as `@cert` (see the section called "`@cert`"). The certainty expressed by `@cert` governs the entire `<ana>`. `<ana>` takes three types of children: `<tok>`, `<joined>`, and `<lm>` (one or more each).

One or more `<tok>`s use the token pattern to point to one or more word tokens or word token fragments (see the section called "`<tok>`"). Unlike uses of `<tok>` in other class 2 files, a TAN-LM `<tok>` is to be treated as strictly a single token. If `@ref`, `@ord`, or `@chars` take multiple values, these are to be treated as constituent parts of a single word token. This feature allows words that have been inadvertently broken up by the tokenization pattern to be rejoined for linguistic analysis. For example, you may wish to treat "pom pom" as a single word token. Under this special definiton of `<tok>` the two parts of that word can be rejoined.

If a particular word or morpheme straddles two adjacent leaf divs, then the references cannot be combined into a single `<tok>`. In those cases, the `<joined>` element should be used to enclose the two or more `<tok>`s that define it.

Both `<joined>` and `<tok>` may take an `@xml:id` value, reserved for use by a future class-2 syntactic format, which will use a TAN-LM file to relate a word token to its larger syntactical relationships.

Each of one or more `<lm>`s collect combinations of lexemes and morphological patterns that are said to be true of the tokens picked. It takes an optional the section called "Edit Stamp" and `@cert` (see the section called "`@cert`"). `<lm>` takes one or more `<l>`s and `<m>`s.

`<l>`, which takes an optional the section called "Edit Stamp" and `@cert` (see the section called "`@cert`"), takes as its content the name of the lexeme (main entry in a lexicon). The optional `@lexicon` may be used to specify a lexicon other than the default one declared in `<body>`. If a lexeme is unattested in the lexicons listed, you may credit yourself as the authority by supplying the appropriate `<agent>` id reference.

The content of `<l>` is a string that points to the main word entry. In many languages, especially those that are lightly inflected, this word will be identical to the word token itself. In those cases, `<l>` may be left empty, indicating that the value of `<tok>` in the `<ana>` must be used. In this case, all values of `<tok>` must resolve to the same value, or a validation error will result. If a lexicon has more than one entry, `@ord` may be used to specify with a number which entry or entries are meant (the section called "`@ord` and `@val`"). If `@ord` is absent, the value is assumed to be 1.

Values of `<l>` point to lexical headwords, not to roots. Root analysis reserved for lexicography, a discipline wholly different from morphological analysis. There is at present no interoperable way to exchange lexicographic data in the TAN format, but the format is set up to allow you, if you wish, to create and develop a proprietary system.

`<m>`, which takes an optional the section called "Edit Stamp" and `@cert` (see the section called "`@cert`"), identifies a morphological pattern that declares the inflectional categories for the word tokens chosen. If a different TAN-R-mor file is intended than the one specified in `<body>`, the optional `@morphology` may be used. Unlike `@lexicon`, you may not credit yourself. If you find a published TAN-R-mor file inadequate, you must use an alternative.

The content of `<m>` consists of one or more codes defined in the chosen TAN-R-mor file. Codes are space-delimited. If a value of `<m>` violates the rules established by the TAN-R-mor file, an error will be generated. For more about how codes are built, and how they function, see the section called "Lexico-Morphology".

As a courtesy to TAN-LM editors, any `<m>` that has too many codes will return not only an error but the meanings of the valid codes currently placed. For example, `<m>rb ?</m>` using a TAN-R-mor file with only one `<category>` might return this error: `Too many codes, which currently resolve: adverb`.

Example 7.32. TAN-LM **`<body>`**

```
  ...........
<body lexicon="LSJ" morphology="Perseus">
  <ana>
     <tok ref="preface 1 title 1" ord="1"/>
     <lm>
        <l lexicon="new">########</l>
        <m>- e - s - - - m g -</m>
     </lm>
  </ana>
  <ana>
     <tok ref="preface 1:title 1" ord="2"/>
     <lm>
        <l>µ######</l>
        <m>n - - s - - - m g -</m>
     </lm>
  </ana>
  <ana>
     <tok ref="preface 1 section 1" ord="1"/>
     <lm>
        <l>####</l>
        <m>c - - - - - - - - -</m>
     </lm>
```

```
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="2"/>
      <lm>
         <l>###</l>
         <m>p p - s - - - - d -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="3"/>
      <lm>
         <l>#####</l>
         <m>d - - - - - - - - -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="4"/>
      <lm>
         <l>#####</l>
         <m>v - 2 s r i a - - -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="5"/>
      <lm>
         <l>###</l>
         <m>r - - - - - - - - -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="6"/>
      <lm>
         <l>#</l>
         <m>p a - s - - - n g -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="7"/>
      <lm>
         <l>#####</l>
         <m>a - - s - - - n g -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="8"/>
      <lm>
         <l>####</l>
         <m>n - - s - - - n g -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="9 | 21"/>
      <lm>
         <l>##</l>
         <m>r - - - - - - - - -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="10"/>
      <lm>
         <l>#</l>
         <m>p a - s - - - f d -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="11"/>
      <lm>
         <l>######</l>
         <m>n p - s - - - f d -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="12"/>
      <lm>
         <l>######µ##</l>
         <m>v - - s p p m m d -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="13"/>
      <lm>
         <l>########</l>
         <m>a - - s - - - m v s</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="14"/>
      <lm>
         <l>#######</l>
         <m>n e - s - - - m v -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="15"/>
      <lm>
         <l>#########</l>
         <m>n e - s - - - m v -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="16"/>
      <lm>
         <l>###</l>
         <m>c - - - - - - - - -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="17"/>
      <lm>
         <l>#</l>
         <m>p a - s - - - n a -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="18"/>
      <lm>
         <l>##µ#######</l>
         <m>a - - s - - - n a -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="19"/>
      <lm>
         <l>###µ#</l>
         <m>n - - s - - - n a -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="20 | 46"/>
      <lm>
         <l>#</l>
         <m>p a - p - - - m g -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="22"/>
      <lm>
         <l>########</l>
         <m>n e - s - - - m d -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="23"/>
      <lm>
         <l>µ######</l>
         <m>n - - p - - - m g -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="24"/>
      <lm>
         <l>###########</l>
         <m>v - - - a n p - - -</m>
      </lm>
   </ana>
   <ana>
      <tok ref="preface 1 section 1" ord="25"/>
      <lm>
         <l>##</l>
         <m>p p - s - - - - d -</m>
      </lm>
   </ana>
```

```
<ana>
   <tok ref="preface 1 section 1" ord="26"/>
   <lm>
      <l>#########</l>
      <m>v - 2 s a i a - - -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="27"/>
   <lm>
      <l>##</l>
      <m>d - - - - - - - - -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="28"/>
   <lm>
      <l>###</l>
      <m>g - - - - - - - - -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="29"/>
   <lm>
      <l>####</l>
      <m>d - - - - - - - - -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="30"/>
   <lm>
      <l>####</l>
      <m>c - - - - - - - - -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="31"/>
   <lm>
      <l>#######</l>
      <m>v - - s p p a n a -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="32"/>
   <lm>
      <l>#####</l>
      <m>p p - s - - - n a -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="33"/>
   <lm>
      <l>##µ###</l>
      <m>v - 2 s r i a - - -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="34"/>
   <lm>
      <l>########</l>
      <m>p d - s - - - f a -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="35"/>
   <lm>
      <l>#########</l>
      <m>n - - s - - - f a -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="36"/>
   <lm>
      <l>###</l>
      <m>v - - s p p a n a -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="37 | 45"/>
   <lm>
      <l>####</l>
      <m>r - - - - - - - - -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="38"/>
   <lm>
      <l>#</l>
      <m>p a - p - - - n a -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="39"/>
   <lm>
      <l>######</l>
      <m>a - - p - - - n a -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="40"/>
   <lm>
      <l>###μ#</l>
      <m>n - - p - - - n a -</m>
   </lm>
</ana>
<ana>
   <tok ref="preface 1 section 1" ord="41"/>
   <lm>
```

```
            <l>#</l>
            <m>p a - p - - - m g -</m>
         </lm>
      </ana>
      <ana>
         <tok ref="preface 1 section 1" ord="42"/>
         <lm>
            <l>########</l>
            <m>n - - p - - - m g -</m>
         </lm>
      </ana>
      <ana>
         <tok ref="preface 1 section 1" ord="43"/>
         <lm>
            <l>####</l>
            <m>v - 2 s p m a - - -</m>
         </lm>
      </ana>
      <ana>
         <tok ref="preface 1 section 1" ord="44"/>
         <lm>
            <l>####</l>
            <m>p r - p - - - n a -</m>
         </lm>
      </ana>
      <ana>
         <tok ref="preface 1 section 1" ord="47"/>
         <lm>
            <l>#####</l>
            <m>a - - p - - - m g -</m>
         </lm>
      </ana>
      <ana>
         <comment who="kalvesmaki" when="2014-02-10">Perseus morph addendum</comme
         <tok ref="preface 1 section 1" ord="48"/>
         <lm>
            <l>#####</l>
            <m>n - - p - - - m g -</m>
         </lm>
      </ana>
      <ana>
         <tok ref="preface 1 section 1" ord="49"/>
         <lm>
            <l>####</l>
            <m>r - - - - - - - - -</m>
         </lm>
      </ana>
      <ana>
         <tok ref="preface 1 section 1" ord="50"/>
         <lm>
            <l>#####</l>
            <m>p d - s - - - n g -</m>
         </lm>
      </ana>
      <ana>
         <tok ref="preface 1 section 1" ord="51"/>
         <lm>
            <l>μ######</l>
            <m>v - 1 p r i a - - -</m>
         </lm>
      </ana>
      <ana>
         <comment who="kalvesmaki" when="2014-02-10">Perseus morph addendum</comme
         <tok ref="preface 1 section 1" ord="52"/>
         <lm>
            <l>#########</l>
            <m>v - 1 p a s a - - -</m>
         </lm>
      </ana>
      ...........
   </body>
      ...........
```

Chapter 8. Class 3 TAN Files: Language and Script

This chapter discusses class 3 TAN files, which pertain to languages and scripts, not to individual texts. This class, at present, consists of rules and patterns pertaining to tokenization and morphology. As the network grows, other types are expected to be added.

Tokenization Patterns

The term *word* is notoriously difficult to define for any language (in English, ”New York” and ”didn't” can each be justifiably defined as one or two words). We adopt instead the proximate term preferred in linguistics, *token*. A token is essentially a word that is defined computationally according to a regular expression, and those regular expressions are constructed such that matches approximate what we normally call words. For example, we may define a token as any continuous string of word characters, `\w+` (see above, the section called “Regular Expressions”).

To *tokenize* is to segment a string into a sequence of tokens. The segmentation is specified by another regular expression that defines the pattern that separates tokens, that is, the opposite or complement set. The way we find the tokens made of `\w+`, our example above, is by tokenizing on the complement set, `\W+`.

Every TAN-R-tok file is, in essence, a tokenization constructor. It describes zero or more preparatory replacement processes, followed by a single tokenizing command, so as to create a sequence of words or tokens that can then be referred to by their position in the sequence. This is the backbone upon which all class-2 files depend for pointing to specific word tokens in one or more class 1 files. Because TAN-R-tok files are used extensively by both class 1 and class 2 files, it is important that you familiarize yourself with how a TAN-R-tok file will be invoked by them. See the section called “Recommended Tokenizations” and the section called “Class 2 Metadata (`<head>`)” [81].

Tokenization files are either language specific or language agnostic. The latter tend to be rather limited in number and size, so for convenience, TAN provides the following language-agnostic TAN-R-tok files as standard rules:

- `http://textalign.net/release/1/dev/rules/TAN-R-tok/general-1.xml`
- `http://textalign.net/release/1/dev/rules/TAN-R-tok/general-words-only-1.xml`
- `http://textalign.net/release/1/dev/rules/TAN-R-tok/precise-1.xml`

For more about these core tokenization patterns and how they are best used, see the files themselves.

Tokenization patterns devoted to specific to languages may be found in the examples on the TAN website.

Principles and Assumptions

Certain assumptions and recommendations are made regarding tokenization files, complementing the more general ones; see the section called “Design Principles”.

TAN-R-tok files are used exclusively to describe how a text should first be changed (if needed) and then tokenized. Editors of these files should be familiar with the writing conventions of the scripts for the languages they are tokenizing and with the behavior of Unicode and Regular Expressions. (See the section called “Unicode” and the section called “Regular Expressions”).

Decisions in tokenizing are motivated by the assumptions one makes about a given language and about the questions that motivates someone to tokenize a text. Some linguists prefer to treat punctuation as tokens in their own right. Others, such as those who work with ancient texts (where much punctuation is erratic, anachronistic, or irrelevant), may wish to suppress them altogether. A poorly chosen or constructed TAN-R-tok file can produce invalid or undesired results. It is of paramount importance that you, as a TAN-R-tok editor, clearly indicate the ideal purposes of the file, both through a description of contents and good examples (at least one of which is required).

Tokenization will be applied to one leaf `<div>` at a time, and not to the concatenation of those `<div>`s. So if a word straddles leaf `<div>`s, it will always be broken up into two or more tokens. There are work-arounds for these situations, but those fixes should be applied in other TAN formats, not this one.

TAN-R-tok files do not admit doubt or multiple opinions.

Rootmost Element and Header

The rootmost element of a tokenizing file is `<TAN-R-tok>`.

One or more optional `<source>` elements may describe previous works that have been used as sources for the patterning rules. Many TAN-R-tok files will have no `<source>` element.

`<declarations>` takes one or more `<for-lang>`s that take a standard language code (see xsd:language). If a TAN-R-tok file is meant to apply to any language, no `<for-lang>` should be supplied.

If another TAN file does not have all the languages specified in a TAN-R-tok file it is using, a warning will be generated. But the warning will not prevent the data from being processed.

Example 8.1. TAN-R-tok **`<head>`**, any language

```
      ...........
   <head>
      <name>Common tokenization pattern no. 1</name>
      <desc>This tokenization pattern takes any text in any language, offsets any
         of non-word characters with spaces, then returns a tokenization based on
         space.</desc>
      <location>tok.common.1.xml</location>
      <rights-excluding-sources rights-holder="kalvesmaki">
         <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
         <name>Creative Commons Attribution 4.0 International License</name>
         <desc>This license is granted independent of rights and licenses associat
            source. </desc>
      </rights-excluding-sources>
      <declarations/>
      <agent xml:id="park" roles="creator">
          <IRI>tag:parkj@textalign.net,2015:self</IRI>
          <name>Jenny Park</name>
      </agent>
      <role xml:id="creator">
         <IRI>http://schema.org/creator</IRI>
         <name xml:lang="eng">creator</name>
      </role>
      <change when="2015-03-16" who="park">Started file</change>
   </head>
      ...........
```

Example 8.2. TAN-R-tok **<head>**, language-specific

```
      ...........
   <head>
      <name>Tokenization pattern for English, in Penn Treebank style</name>
      <desc>This tokenization pattern is designed to imitate the tokenization proc
         the Penn Treebank project on modern English texts. Expected input: raw te
         at a time.</desc>
      <location>tok.eng.1.xml</location>
      <rights-excluding-sources rights-holder="kalvesmaki">
         <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
         <name>Creative Commons Attribution 4.0 International License</name>
         <desc>This license is granted independent of rights and licenses associat
            source. </desc>
      </rights-excluding-sources>
      <see-also>
         <relationship>auxiliary</relationship>
         <IRI>tag:parkj@textalign.net,2015:auxiliary:tokenizing-sed-script-1</IRI>
         <name>sed tokenizing script, by Robert MacIntyre</name>
         <desc>Written ca. 1995 at the University of Pennsylvania</desc>
         <location when-accessed="2015-03-17"
            >http://www.cis.upenn.edu/~treebank/tokenizer.sed</location>
      </see-also>
      <declarations>
         <for-lang>en</for-lang>
         <for-lang>eng</for-lang>
      </declarations>
      <agent xml:id="park" roles="creator">
          <IRI>tag:parkj@textalign.net,2015:self</IRI>
          <name>Jenny Park</name>
      </agent>
      <role xml:id="creator">
         <IRI>http://schema.org/creator</IRI>
         <name xml:lang="eng">creator</name>
      </role>
      <change when="2015-03-17" who="park">Started file</change>
   </head>
      ...........
```

Data (<body>)

Table 8.1. Synopsis of TAN-R-tok **<body>**, **<replace>**, **<tokenize>**, **<pattern>**, **<replacement>**, **<flags>**, **<example>**, **<input>**, **<output-token>**

name	req?	qty	parent	attributes	children	purpose
`<body>`	yes	one	`<TAN-R-tok>`	the section called "@in-progress"?, the section called "Edit Stamp"?	`<replace>`*, `<tokenize>`, `<example>`+	contains data

name	req?	qty	parent	attributes	children	purpose
`<replace>`	no	one or more	`<body>`	`@xpath`?, the section called “Edit Stamp”?	`<pattern>`, `<replacement>`, `<flags>`?	state a pattern to be searched and replaced in a digital source
`<tokenize>`	yes	one	`<body>`	the section called “Edit Stamp”?	`<pattern>`, `<flags>`?	declares tokenization function to be applied to a string
`<pattern>`	yes	one	`<replace>`, `<tokenize>`	the section called “Edit Stamp”?	data (xsd:string)	state through a regular expression a pattern to be found
`<replacement>`	yes	one	`<replace>`	the section called “Edit Stamp”?	data (xsd:string)	state through a regular expression a replacement pattern
`<flags>`	no	one	`<replace>`, `<tokenize>`	the section called “Edit Stamp”?	data (xsd:string)	declare flags to be applied in a replace operation
`<example>`	yes	one or more	`<body>`	the section called “Edit Stamp”?	`<input>`, `<output-token>`+	Provide an example of a tokenized string
`<input>`	yes	one	`<example>`	the section called “Edit Stamp”?	data (xsd:string)	Provide example input
`<output-token>`	yes	one or more	`<example>`	the section called “Edit Stamp”?	data (xsd:string)	Provide example output token

The `<body>` of a TAN-R-tok file takes the customary optional attributes found in other TAN files (see the section called “@in-progress” and the section called “Edit Stamp”). `<body>` has only three kinds of children: zero or more `<replace>`s followed by one `<tokenize>` followed by one or more `<example>`s .

`<replace>` corresponds to `fn:replace` as defined in XPath Functions 3.0 [http://www.w3.org/TR/xpath-functions-30/#func-replace]. `<replace>` may take an optional `@xpath` value, to restrict the scope of the replacement to a specific place in a valid XML source.

Caution

Using `@xpath` is not recommended if you anticipate users who may be relying upon XSLT 2.0 and earlier for transformations. Only from version 3.0 has XPath been able to convert a string datatype (the datatype of `@xpath`) into an XPath expression.

`<replace>` must take one `<pattern>` and one `<replacement>`, two regular expressions (see the section called “Regular Expressions”) to be fed as parameters into an XPath `fn:replace` function. It may take an optional `<flags>`, which adds parameters to the search, i.e., any combination of `<s>` (”dot-all” mode), `<m>` (multi-line mode), `<i>` (case-insensitive mode), `<x>` (strip white spaces from `<pattern>`), or `<q>` (suppress metacharacters) as values, explained in greater detail in the W3C recommendation [http://www.w3.org/TR/xpath-functions-30/#flags].

Multiple `<replace>` values will be applied according to document order. It is very important to be careful to anticipate ways an early search might adversely affect a later one.

Each `<tokenize>` has exactly one `<pattern>` element and an optional `<flags>` element. The values of these two elements, both strings, represent regular expressions that should be used as parameters for `fn:tokenize` as defined in XPath Functions 3.0 [http://www.w3.org/TR/xpath-functions-30/#func-tokenize]. The value of `<pattern>` is a regular expression that will be used to segment a leaf `<div>` into a sequence of tokens; `<flags>` takes the same options `<replace>` does.

Finally, one or more `<example>` elements provide one `<input>` and one or more `<output-token>`s. `<input>` takes any string, and the resultant sequence of tokens is supplied in `<output-token>`. If `<input>`, when fed into the sequence of `<replace>`s and `<tokenize>`, does not produce the sequence declared by the `<output-token>`s, an error will be generated.

Example 8.3. TAN-R-tok **`<body>`**, treating punctuation clusters as word tokens

```
   ...........
<body>
   <replace>
      <comment when="2015-03-16" who="park">Pad every cluster of non-word chara
         space.</comment>
      <pattern>(\W+)</pattern>
      <replacement> $1 </replacement>
   </replace>
   <replace>
      <comment when="2015-03-16" who="park">Delete end spaces.</comment>
      <pattern>^\s+|\s+$</pattern>
      <replacement/>
   </replace>
   <tokenize>
      <pattern>\s+</pattern>
   </tokenize>
   <example>
      <input>I said, "Where is the ping-pong table?"</input>
      <output-token>I</output-token>
      <output-token>said</output-token>
      <output-token>,</output-token>
      <output-token>"</output-token>
      <output-token>Where</output-token>
      <output-token>is</output-token>
      <output-token>the</output-token>
      <output-token>ping</output-token>
      <output-token>-</output-token>
      <output-token>pong</output-token>
      <output-token>table</output-token>
      <output-token>?"</output-token>
   </example>
</body>
   ...........
```

Example 8.4. TAN-R-tok **<body>**, ignoring punctuation

```
   ...........
<body>
   <replace>
      <comment when="2015-03-16" who="park">Remove end nonword characters</comm
      <pattern>^\W+|\W+$</pattern>
      <replacement/>
   </replace>
   <tokenize>
      <comment when="2015-03-16" who="park">Tokenize on any cluster of nonword
         characters.</comment>
      <pattern>\W+</pattern>
   </tokenize>
   <example>
      <input>I said, "Where is the ping-pong table?"</input>
      <output-token>I</output-token>
      <output-token>said</output-token>
      <output-token>Where</output-token>
      <output-token>is</output-token>
      <output-token>the</output-token>
      <output-token>ping</output-token>
      <output-token>pong</output-token>
      <output-token>table</output-token>
   </example>
</body>
   ...........
```

Example 8.5. TAN-R-tok **<body>**, tokenizing English in the manner of the Penn Treebank project

```
   ...........
<body>
   <replace>
      <comment when="2015-03-17" who="park">get directional quotes</comment>
      <pattern>^"</pattern>
      <replacement>`` </replacement>
   </replace>
   <replace>
      <pattern>([ \(\[{&lt;])"</pattern>
      <replacement>$1 `` </replacement>
   </replace>
   <comment when="2015-03-17" who="park">close quotes at end</comment>
   <replace>
      <pattern>\.\.\.</pattern>
      <replacement>...</replacement>
   </replace>
   <replace>
      <pattern>([,;:@#$%&amp;])</pattern>
      <replacement> $1 </replacement>
   </replace>
   <replace>
      <comment when="2015-03-17" who="park">Offset final periods</comment>
      <pattern>([^.])([.])([\]\)}>"']*)$</pattern>
      <replacement>$1 $2$3</replacement>
   </replace>
   <replace>
      <comment when="2015-03-17" who="park">Offset question marks, exclamation
         punctuation</comment>
      <pattern>([?!\]\[\(\){}&lt;>—–])</pattern>
      <replacement> $1 </replacement>
   </replace>
   <replace>
      <comment when="2015-03-17" who="park">Offset double hyphens</comment>
      <pattern>--</pattern>
      <replacement> $1 </replacement>
   </replace>
   <replace>
      <comment when="2015-03-17" who="park">Insert opening space, to reduce reg
         searches.</comment>
      <pattern>^(.)</pattern>
      <replacement> $1</replacement>
   </replace>
   <replace>
      <comment when="2015-03-17" who="park">Insert closing space.</comment>
      <pattern>(.)$</pattern>
      <replacement>$1 </replacement>
   </replace>
   <replace>
      <comment when="2015-03-17" who="park">Offset closing quotation marks</com
      <pattern>"</pattern>
      <replacement> '' </replacement>
   </replace>
   <replace>
      <comment when="2015-03-17" who="park">Possessive or close single quote</c
      <pattern>([^'])' </pattern>
      <replacement>$1 ' </replacement>
   </replace>
   <replace>
      <comment when="2015-03-17" who="park">Separate contracted enclitics, e.g.
         I'm</comment>
      <pattern>'([sSmMdD]) </pattern>
      <replacement> '$1 </replacement>
   </replace>
   <replace>
      <comment when="2015-03-17" who="park">'ll, 're, 've</comment>
      <pattern>'(ll|re|ve) </pattern>
      <replacement> '$1 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <comment when="2015-03-17" who="park">n't</comment>
      <pattern>(n't) </pattern>
      <replacement> $1 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> (can)(not) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> (d')(ye) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> (gim)(me) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> (gon)(na) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> (got)(ta) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> (lem)(me) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> (more)('n) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> ('t)(is) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> ('t)(was) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> (wan)(na) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
      <pattern> (whad)(dya) </pattern>
      <replacement> $1 $2 </replacement>
      <flags>i</flags>
   </replace>
   <replace>
```

```
            <pattern> (wha)(t)(cha) </pattern>
            <replacement> $1 $2 $3 </replacement>
            <flags>i</flags>
         </replace>
         <replace>
            <comment when="2015-03-17" who="park">Strip opening and closing space.</c
            <pattern>(^\s+|\s+$)</pattern>
            <replacement/>
         </replace>
         <tokenize>
            <pattern>\s+</pattern>
         </tokenize>
         <example>
            <input>"I said, ["Wanna play ping-pong?"></input>
            <output-token>``</output-token>
            <output-token>I</output-token>
            <output-token>said</output-token>
            <output-token>,</output-token>
            <output-token>[</output-token>
            <output-token>``</output-token>
            <output-token>Wan</output-token>
            <output-token>na</output-token>
            <output-token>play</output-token>
            <output-token>ping-pong</output-token>
            <output-token>?</output-token>
            <output-token>''</output-token>
            <output-token>></output-token>
         </example>
      </body>
         ...........
```

Morphological Concepts and Patterns (TAN-R-mor)

TAN-R-mor files describe the morphological categories for a given language and the rules allowed in applying them to any given word token. The format allows specificity, flexibility, and responsiveness. Assertions in the format may be doubted, rules may be expressed as contingent upon other coniditons, and warnings and error messages may be sent to users who may have used a pattern incorrectly, or not in with the best practices.

The TAN-R-mor format is, in essence, like a Schematron language for languages. That is, you may specify series of reports and assertions to be made, based upon how user of the format tries to combine morphological features. So a user is not only able to see if the rules have been violated, but why, and exactly where.

This chapter should be read in close conjunction with that pertaining to TAN-LM files, which exclusively depend upon them (the section called "Lexico-Morphology").

Principles and Assumptions

Certain assumptions and recommendations are made regarding morphology files, complementing the more general ones; see the section called "Design Principles".

TAN-R-mor files are restricted exclusively to the description of the inflectional categories that characterize a language, and to declare rules that restrict how those categories can be declared or combined. Editors of these files should be familiar with the grammar of the languages they are describing.

The TAN-R-mor format has been designed under the assumption that word formation and inflection, in any given language, can be analyzed in different ways. It is also assumed that patterns of word inflection and formation can be categorized, classified, named, and described. Even if two views of a single language are in disagreement, those disagreements should be stated in comparable terms, when possible.

For example, not everyone agrees on the number of major parts of speech in English. And among those who think there are only eight, some name and define those eight categories in ways that are at odds the consensus of modern linguists. The older paradigm held to a major category called *conjunctions*, whereas most linguists prefer to break this into two major categories, *subordinators* and *coordinators*.

The TAN-R-mor format has also been designed to cater to two approaches to coding the morphological features of a language: structured or unstructured.

Structured codes are created with a presumption of a set number of categories into which various features of morphology should be combined. Structured codes are of a set length, and usually require gaps in different spots. For example, the Perseus approach to the morphological categories of Greek, Latin, and other highly inflected languages dictate ten categories, with the first two being the major and minor parts of speech, and the subsequent categories devoted to person, number, tense, and so forth. For each word being analyzed, all ten categories must have a value, even if null.

Morphology based on unstructured codes relies upon a single master set of tags for morphological features. Word tokens in that language can be given one or more tags drawn from that set. This approach is viable for any language, but it is most often found in tagging sets for languages that have little inflection, e.g., the Brown and Penn sets for English.

Both structured and unstructured approaches begin the same in every TAN-R-mor file: a set of all possible morphological features, in the `<declarations>` part of the header. If one wishes, this set can then be assigned a series of categories, to turn the unstructured tag set into a structured one. This process is described below.

Rootmost Element and Header

The rootmost element of a morphological rule file is `<TAN-R-mor>`.

Zero or more `<source>` elements describe the grammars or related works that account for the rules declared in the TAN file. If the rules are not based upon any published work, then `<source>` may be omitted. Any TAN-R-mor file without a source will assume to be based upon the personal knowledge of the `<agent>`s who edited the file.

`<declarations>` takes two types of element, one or more `<for-lang>`s that take a standard language code (see xsd:language) and one or more `<feature>`s that declare the morphological features that characterize the languages being described.

<I left off at this point. This entire section needs to be rewritten in light of the new TAN-R-mor schemas.>

Example 8.6. TAN-R-mor **`<head>`**

```
    ...........
  <head>
    <name>Perseus rules for Greek morphology</name>
```

```
      <rights-excluding-sources rights-holder="kalvesmaki">
         <IRI>http://creativecommons.org/licenses/by/4.0/deed.en_US</IRI>
         <name>Creative Commons Attribution 4.0 International License</name>
         <name>This license is granted independent of rights and licenses associat
            source. </name>
      </rights-excluding-sources>
      <source>
         <IRI>urn:uuid:147d8040-4ca9-11e3-848f-0002a5d5c51b:2013-11-18</IRI>
         <name>Perseus codes</name>
         <desc xml:lang="eng">Perseus's standard code for Greek morphology (a stri
            found at http://perseus.uchicago.edu/about.html (accessed 2013-11-18)<
      </source>
      <declarations>
         <for-lang>grc</for-lang>
      </declarations>
      <agent xml:id="kalvesmaki" roles="editor">
         <IRI>http://viaf.org/viaf/299582703</IRI>
         <IRI>tag:kalvesmaki@gmail.com,2014:self</IRI>
         <name>Joel Kalvesmaki</name>
      </agent>
      <role xml:id="editor">
         <IRI>http://schema.org/editor</IRI>
         <name>Editor</name>
      </role>
      <change who="kalvesmaki" when="2014-02-26">File created</change>
      <comment when="2013-11-16" who="kalvesmaki">This was written to codify for t
         the 10-character codes developed by Perseus for ancient Greek. </comment>
   </head>
      ...........
```

Data (<body>)

Table 8.2. Synopsis of TAN-R-mor **`<body>`**, **`<invalid-pattern>`**, **`<category>`**, **`<option>`**

name	req?	qty	parent	attributes	children	purpose
`<body>`	yes	one	`<TAN-R-mor>`	the section called "@in-progress"?, the section called "Edit Stamp"?	`<invalid-pattern>`*, `<category>`+	contains data
`<invalid-pattern>`	no	one or more	`<body>`	the section called "Edit Stamp"?, `@cert`?	text (xsd:string)	declares a pattern of space-delimited codes disallowed
`<category>`	yes	one or more	`<body>`	the section called "Edit Stamp"?	the section called "IRI + name Pattern", `<option>`+	collects a group of morphological concepts
`<option>`	yes	one or more	`<category>`	the section called "Edit Stamp"?, `@code`	the section called "IRI +	names a morphological concept that is

name	req?	qty	parent	attributes	children	purpose
					name Pattern"	a viable option for the given category

The `<body>` of a TAN-R-mor file takes the customary optional attributes found in other TAN files (see the section called "@in-progress" and the section called "Edit Stamp"). `<body>` has only two type of children: zero or more `<invalid-pattern>`s followed by one or more `<category>`s followed by zero or more `<invalid-pattern>`s .

`<invalid-pattern>` takes a regular expression defining patterns of codes that are disallowed (explained below). Aside from an optional the section called "Edit Stamp" and `@cert`?, it takes only text, followed perhaps by a single `<comment>`. `@cert` takes a numeral from 0 to 1 or else `<high>` or `<low>`, which are to be translated to mean "is probably" and "is probably not" respectively.

`<category>`, in addition to an optional the section called "Edit Stamp", takes the the section called "IRI + name Pattern" to describe a category of morphological terms, and one or more `<option>`s.

`<option>`, which describes a single morphological concept, takes an the section called "Edit Stamp" and the the section called "IRI + name Pattern". It also takes a mandatory `@code`, which must be unique within a given category. But that value may be used in other `<category>`s. `@code` may by anything that is not a space character or a hyphen (reserved for the concept "not applicable").

There is no particular rule as to what concepts are allowed in a `<category>`. In highly inflected languages, it makes sense to categorize according to part of speech. For languages with little inflection, a single `<category>` might contain all the options available.

The TAN-R-mor file will be referred to by TAN-LM files by using one or more `@code` values, space delimited, with each value corresponding in order to the `<category>` order. For example, if a Greek TAN-R-mor file is assigned ten categories, then any TAN-LM file using that file will have a value that looks like `x x x x x x x x x x`, where `<x>` is a code value. If a particular category is not applicable, the hyphen is used. Anyone using the TAN-R-mor file will not be able to cite more than one `<option>` in a given `<category>`.

`@code` values may rendered in any case you wish, but they must not be treated as case-sensitive. If within any category one `@code` is a case variant of another (e.g., `<S>` and `<s>`), then the file will be marked as invalid.

Every `<option>` must refer to a morphological concept that is unique within the entire file. That is, if you wish to allow the concept *adverb* as an option, that `<option>` must appear in only one `<category>`, and no others.

Collectively, all the `@code` and `<category>`s provide the maximal limits of what is allowed. `<invalid-pattern>` allows you to restrict those limits to what is permitted in standard grammatical analysis. Let us take for example a TAN-R-mor file that defines the term *conjunction* in the first `<category>` and *first-person* in the second, and let us suppose the language has no concept of a first-person conjunction. You may disallow this combination in an `<invalid-pattern>` with the pattern `^c 1`, where `<c>` and `<1>` are the codes for the two concepts. (The `^` is a standard regular expression anchor, indicating that `<c>` should be at the beginning of the searched string.) These patterns may be combined in accordance with regular expressions (see the section called "Regular Expressions"). They make the presumption that the string has had its space normalized, so `\s+` ("one or more space characters") is not necessary in an `<invalid-pattern>`.

If you are editing a TAN-R-mor file that has already been published, and you are adding or altering the pattern invalidity restrictions. In this case, it is a good idea to make sure that `@in-progress` is removed or set to false, and to add new `<invalid-pattern>`s with the `@cert` set to a value other than 1. Doing so means that anyone using the old version will not have their data rendered invalid, and it will alert them both to new rules that have been included. At some point, after editing is complete you will be prepared to declare the morphology file to be a new version altogether. At that point, change the IRI name of the file, remove the `@cert` values that were placeholders, and save the file as a different file name. Do not remove the older, published version, since users may be depending upon it.

Although it is good practice to declare as many invalid patterns as you can, you are not required to state every possible combination that is not allowed. If you are not confident that a given pattern should be disallowed, use the `@cert` value in `<invalid-pattern>`. Any TAN-LM file using that value will not be marked as invalid, but a warning will be delivered to the user, along with the content of any `<comment>` that is included.

Because an example is the best way to illustrate how one might structure categories in a TAN-R-mor `<body>`, a full `<example>`, but with

Example 8.7. TAN-R-mor **<body>**

```
      ...........
   <body>
      <category>
         <IRI>http://dbpedia.org/resource/Part_of_speech</IRI>
         <name>major part of speech</name>
         <option code="a">
            <IRI>http://dbpedia.org/resource/Adjective</IRI>
            <name>adjective</name>
         </option>
         <option code="c">
            <IRI>http://dbpedia.org/resource/Conjunction</IRI>
            <name>conjunction</name>
         </option>
         <option code="d">
            <IRI>http://dbpedia.org/resource/Adverb</IRI>
            <name>adverb</name>
         </option>
         <option code="e">
            <IRI>tag:textalign.net,2015:morphology:concept:#vocative</IRI>
            <name># vocative</name>
         </option>
         <option code="g">
            <IRI>http://dbpedia.org/resource/Grammatical_particle</IRI>
            <name>particle</name>
         </option>
         <option code="i">
            <IRI>http://dbpedia.org/resource/Interjection</IRI>
            <name>interjection</name>
         </option>
         <option code="m">
            <IRI>http://dbpedia.org/resource/Numeral_(linguistics)</IRI>
            <name>numeral</name>
         </option>
         <option code="n">
            <IRI>http://dbpedia.org/resource/Noun</IRI>
            <name>noun</name>
         </option>
         <option code="p">
            <IRI>http://dbpedia.org/resource/Pronoun</IRI>
            <name>pronoun</name>
         </option>
         <option code="r">
            <IRI>http://dbpedia.org/resource/Preposition</IRI>
            <name>preposition</name>
         </option>
         <option code="v">
            <IRI>http://dbpedia.org/resource/Verb</IRI>
            <name>verb</name>
         </option>
         <option code="y">
            <IRI>http://dbpedia.org/resource/Acronym</IRI>
            <name>acronym</name>
         </option>
      </category>
      <category>
         <IRI>urn:uuid:40aa0be5-6746-4f39-b70f-6c94fd1e4578 </IRI>
         <name>minor part of speech</name>
         <option code="a">
            <IRI>http://dbpedia.org/resource/Article_(grammar)</IRI>
            <name>article or determinative</name>
         </option>
         <option code="c">
            <IRI>http://dbpedia.org/resource/Reciprocal_pronoun</IRI>
            <name>reciprocal pronoun</name>
         </option>
         <option code="d">
            <IRI>http://dbpedia.org/resource/Demonstrative</IRI>
            <name>demonstrative</name>
         </option>
         <option code="e">
            <IRI>http://dbpedia.org/resource/Proper_noun</IRI>
            <name>proper</name>
         </option>
         <option code="i">
            <IRI>http://dbpedia.org/resource/Interrogative</IRI>
            <name>interrogative</name>
         </option>
         <option code="k">
            <IRI>http://dbpedia.org/resource/Reflexive_pronoun</IRI>
            <name>reflexive</name>
         </option>
         <option code="m">
            <IRI>tag:textalign.net,2015:morphology:concept:modal</IRI>
            <name>modal</name>
            <name>modal (particle)</name>
         </option>
         <option code="p">
            <IRI>http://dbpedia.org/resource/Personal_pronoun</IRI>
            <name>personal pronoun</name>
         </option>
         <option code="r">
            <IRI>http://dbpedia.org/resource/Relative_pronoun</IRI>
            <name>relative pronoun</name>
         </option>
         <option code="s">
            <IRI>http://dbpedia.org/resource/Possessive_pronoun</IRI>
            <name>possessive pronoun</name>
         </option>
         <option code="v">
            <IRI>tag:textalign.net,2015:morphology:concept:verbal</IRI>
            <name>verbal</name>
         </option>
         <option code="x">
            <IRI>http://dbpedia.org/resource/Indefinite_pronoun</IRI>
            <name>indefinite</name>
         </option>
      </category>
      <category>
         <IRI>http://dbpedia.org/resource/Grammatical_person</IRI>
         <name>person</name>
         <option code="1">
            <IRI>urn:uuid:754589f6-8f64-11e3-950a-425861b86ab6</IRI>
            <name>first person</name>
         </option>
         <option code="2">
            <IRI>urn:uuid:75458d20-8f64-11e3-950a-425861b86ab6</IRI>
            <name>second person</name>
         </option>
         <option code="3">
            <IRI>urn:uuid:75459068-8f64-11e3-950a-425861b86ab6</IRI>
            <name>third person</name>
         </option>
      </category>
      <category>
         <IRI>http://dbpedia.org/resource/Grammatical_number</IRI>
         <name>number</name>
         <option code="d">
            <IRI>http://dbpedia.org/resource/Dual_(grammatical_number)</IRI>
            <name>dual</name>
         </option>
         <option code="p">
            <IRI>http://dbpedia.org/resource/Plural</IRI>
            <name>plural</name>
         </option>
         <option code="s">
            <IRI>http://dbpedia.org/resource/Singulative_number</IRI>
            <name>singular</name>
         </option>
      </category>
      <category>
         <IRI>http://dbpedia.org/resource/Grammatical_tense</IRI>
         <name>tense</name>
         <option code="a">
            <IRI>http://dbpedia.org/resource/Aorist</IRI>
            <name>aorist</name>
         </option>
         <option code="f">
            <IRI>http://dbpedia.org/resource/Future_tense</IRI>
            <name>future</name>
         </option>
         <option code="i">
            <IRI>http://dbpedia.org/resource/Imperfect_tense</IRI>
            <name>imperfect</name>
         </option>
         <option code="l">
            <IRI>http://dbpedia.org/resource/Pluperfect</IRI>
            <name>pluperfect</name>
         </option>
         <option code="p">
            <IRI>http://dbpedia.org/resource/Present_tense</IRI>
            <name>present</name>
         </option>
         <option code="r">
            <IRI>http://dbpedia.org/resource/Perfect_tense</IRI>
            <name>perfect</name>
         </option>
         <option code="t">
            <IRI>http://dbpedia.org/resource/Future_perfect</IRI>
            <name>future perfect</name>
         </option>
      </category>
      <category>
         <IRI>http://dbpedia.org/resource/Grammatical_mood</IRI>
         <name>mood</name>
         <option code="i">
            <IRI>http://dbpedia.org/resource/Realis_mood</IRI>
            <name>indicative</name>
         </option>
         <option code="m">
            <IRI>http://dbpedia.org/resource/Imperative_mood</IRI>
            <name>imperative</name>
         </option>
         <option code="n">
            <IRI>http://dbpedia.org/resource/Infinitive</IRI>
            <name>infinitive</name>
         </option>
         <option code="o">
            <IRI>http://dbpedia.org/resource/Optative_mood</IRI>
            <name>optative</name>
         </option>
         <option code="p">
            <IRI>http://dbpedia.org/resource/Participle</IRI>
            <name>participle</name>
         </option>
         <option code="s">
            <IRI>http://dbpedia.org/resource/Subjunctive_mood</IRI>
            <name>subjunctive</name>
```

```
   </option>
</category>
<category>
   <IRI>http://dbpedia.org/resource/Voice_(grammar)</IRI>
   <name>voice</name>
   <option code="a">
      <IRI>http://dbpedia.org/resource/Active_voice</IRI>
      <name>active</name>
   </option>
   <option code="e">
      <IRI>http://dbpedia.org/resource/Mediopassive_voice</IRI>
      <name>middle-passive</name>
   </option>
   <option code="m">
      <IRI>http://dbpedia.org/resource/Middle_voice#Middle</IRI>
      <name>middle</name>
   </option>
   <option code="p">
      <IRI>http://dbpedia.org/resource/Passive_voice</IRI>
      <name>passive</name>
   </option>
</category>
<category>
   <IRI>http://dbpedia.org/resource/Grammatical_gender</IRI>
   <name>gender</name>
   <option code="c">
      <IRI>http://dbpedia.org/resource/Common_gender</IRI>
      <name>common</name>
   </option>
   <option code="f">
      <IRI>http://dbpedia.org/resource/Feminine_gender</IRI>
      <name>feminine</name>
   </option>
   <option code="m">
      <IRI>http://dbpedia.org/resource/Masculine_gender</IRI>
      <name>masculine</name>
   </option>
   <option code="n">
      <IRI>http://dbpedia.org/resource/Neuter_gender</IRI>
      <name>neuter</name>
   </option>
</category>
<category>
   <IRI>http://dbpedia.org/resource/Grammatical_case</IRI>
   <name>case</name>
   <option code="a">
      <IRI>http://dbpedia.org/resource/Accusative</IRI>
      <name>accusative</name>
   </option>
   <option code="d">
      <IRI>http://dbpedia.org/resource/Dative</IRI>
      <name>dative</name>
   </option>
   <option code="g">
```

```
               <IRI>http://dbpedia.org/resource/Genitive</IRI>
               <name>genitive</name>
            </option>
            <option code="n">
               <IRI>http://dbpedia.org/resource/Nominative</IRI>
               <name>nominative</name>
            </option>
            <option code="v">
               <IRI>http://dbpedia.org/resource/Vocative</IRI>
               <name>vocative</name>
            </option>
         </category>
         <category>
            <IRI>http://dbpedia.org/resource/Comparison_(grammar)</IRI>
            <name>degree</name>
            <option code="c">
               <IRI>http://dbpedia.org/resource/Comparative</IRI>
               <name>comparative</name>
            </option>
            <option code="s">
               <IRI>http://dbpedia.org/resource/Superlative</IRI>
               <name>superlative</name>
            </option>
         </category>
         <invalid-pattern cert="high">^e</invalid-pattern>
         <comment who="kalvesmaki" when="2015-02-28">Code e has been deprecated. It w
            cases for # voc., to be distinguished from # as exclamation </comment>
         <invalid-pattern cert="high">^. v</invalid-pattern>
         <comment who="kalvesmaki" when="2015-02-28">Code v for verbal adjective has
         <invalid-pattern>^c[^ -]</invalid-pattern>
         <comment who="kalvesmaki" when="2015-02-28">A conjunction has no other infle
      </body>
         ...........
```

<caption>

This example provides one morphological analysis of the Greek language. It furnishes only three of many possible `<invalid-pattern>`s, to keep the example brief. Any TAN-LM editor that uses in a morphological analysis will be warned or find their file is rendered invalid when: (1) `<e>` is the first value (warning only); (2) `<v>` is the second value (warning only); (3) `<c>` is the first value and is followed by other values, except for the hyphens (i.e., not applicable) (fully invalid). In each case, the first comment that follows the `<invalid-pattern>` will be delivered to the editor.
</caption>

Part III. Working with the Text Alignment Network

Table of Contents

Chapter 9. Best Practices in Creating and Editing TAN Files

❗ Draft Boundary

AUTHOR SUSPENDED WRITING HERE; THIS POINT FORWARD IS INCOMPLETE

Filenames: Good idea to give core TAN files a pre-extension corresponding to their file type. For example, a project on Shakespeare's Hamlet converts their TEI files, WS_Hamlet_eng.xml and WS_Hamlet_fra.xml to TAN-T files and so names them WS_Hamlet_eng.TAN-T.xml and WS_Hamlet_fra.TAN-T.xml. Morphological files of these two might be called WS_Hamlet_eng.TAN-LM.xml and WS_Hamlet_fra.TAN-LM.xml. An alignment file that joins the two TAN-T files might be called WS_Hamlet_eng_fra.TAN-A.xml (or maybe WS_Hamlet_eng.TAN-A.WS_Hamlet_fra.xml).

Chapter 10. Best Practices in Using TAN Files

Developers: it is advisable to process the data in two steps. The first step puts all the data into a simplified, normalized data set. The second step queries and arranges that simplified data.

The first step: collect all the files that have been invoked. Normalize the data according to the `<declaration>`s specified in the `<head>`. Transcription references should be homogenized to have a common nomenclature. It may help to flatten the hierarchy of the transcription data into leaf `<div>`s, i.e., pairs consisting of a leaf `<div>` ref and the text values for that ref.

The second step, that of transforming or querying the text, becomes much simpler and easier.