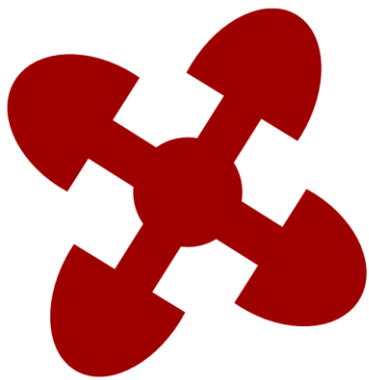
The Pocket Guide to
Akoma Ntoso

2nd Edition

The Pocket Guide to
Akoma Ntoso

2nd Edition

Grant Vergottini



Xcential
LEGISLATIVE TECHNOLOGIES

The Pocket Guide to Akoma Ntoso

By Grant Vergottini

Printing History:
Second Edition: April 2018

10 9 8 7 6 5 4 3 2

841 Second Street
Encinitas, California 92024
http://xcential.com

Contents

Akoma Ntoso Pocket Guide

Preface

I've been working on the OASIS LegalDocML technical committee for over five years, refining the Akoma Ntoso specification for Legislative XML. We have finally arrived at the first release of this specification as an official OASIS standard. It's been a long journey, much longer than any of us ever anticipated.

In my travels, I'm getting one question over and over – 'Where do I start?'

Compared to many schemas I've looked at, Akoma Ntoso isn't all that complicated. However, it is the vast sprawling sea of tags and that seems to be everyone's first stumbling block. Knowing what parts to focus on takes some experience and not everyone has a clear enough vision of the whole to be able to see what parts aren't important to their use cases.

I've been asked so many time to provide an overview of Akoma Ntoso that I thought it would be a good time to publish a concise book and a series of blogs so that everyone might have the opportunity to get an Akoma Ntoso primer.

This primer is a broad overview of Akoma Ntoso, touching on all the subjects that I have found to be useful. As a result, this primer is, by no means, a normative description of Akoma Ntoso and, while it has been reviewed by the LegalDocML technical committee, it is a publication of Xcential Corporation.

I've focused on the parts of Akoma Ntoso that I have experience with rather than try to be comprehensive by covering topics that I've never touched on in my own application of Akoma Ntoso.

As a software developer rather than a legal practitioner, I have focused my primer on the technical aspects of the schema rather than any legal ramifications.

In the first chapter, I'm going to provide an overview of Akoma Ntoso — introducing the schema, discussing its history, motivations, and how best to start to tackle the schema.

For a more comprehensive treatment of Akoma Ntoso, the following resources are available:

- The Official OASIS website: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=legaldocml
- The GitHub repository: https://github.com/oasis-open/legaldocml-akomantoso
- The AkomaNtoso website: http://akomantoso.org

This book was revised by the LegalDocML OASIS technical committee.

Akoma Ntoso Pocket Guide

Chapter 1: An Overview

What is Akoma Ntoso?

Akoma Ntoso is an XML schema for modelling the structure of most documents found in most legislative processes.

Akoma Ntoso began as an initiative at the United Nations Department of Economic and Social Affairs in Kenya in 2006. Since then, it has been developed through the stewardship of Monica Palmirani and Fabio Vitali at the University of Bologna. For the past five years, it has been refined to become the Akoma Ntoso standard inside of the LegalDocML Technical Committee of the OASIS standards body.

True to its African heritage, Akoma Ntoso's name also has an African origin. It means *linked hearts* in the West African Akan language – embodying agreement and unity among families and community. The adinkra[1] symbol for Akoma Ntoso depicts four hearts, linked together as shown:

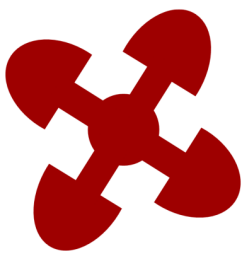


Figure 1 The four linked hearts of Akoma Ntoso

While the name looks a bit daunting, it is pronounced simply as '*a-coma-in-toso*'.

Why is it important?

Akoma Ntoso provides a common foundation for building rulemaking systems, including solutions for regulations and legislation, which enables developers to share experiences and know-how. I am hopeful it

[1] Visual symbols, created by the Ashanti people, that represent concepts or principles.

will spur the growth of an ecosystem of interoperable applications.

Building legislative systems is very expensive and has proven to be quite a risky endeavour. For every success story, there have been many costly failures, leaving many jurisdictions with antiquated methodologies and an urgent need to modernize.

Akoma Ntoso is more than just an XML schema, it's a worldwide community of practitioners of the art of building these systems. It is our hope that, as Akoma Ntoso gains acceptance, this community will grow and we will be able to dramatically lower both the cost and risk of building legislative systems.

Why is it based on XML?

XML provides the best platform for the future — providing a neutral basis for automation that can be implemented with various technologies. Unlike proprietary data formats which favour one vendor's tools over all others, XML data provides a level playing field for a whole ecosystem of interrelated applications. Also, while proprietary data formats tend to be tied to a particular generation of tools, open XML data can be expected to last for many decades, if not longer.

While a case is often made to use word-processing technologies or relational database technologies instead of XML, it is XML that provides the best compromise. XML allows user-friendly drafting tools to be built while also allowing for the best automation of downstream processes such as the amendment process. As a neutral standard, XML future-proofs the data while technologies evolve.

Aren't all legislative processes unique?

Yes, and no. While it is certainly true that no two jurisdictions share the exact same process, when you dig a bit deeper and focus on the documents rather than the process, you find surprising similarities throughout the world. Quite often, the differences are more related to terminology and decoration. It is the commonality that we have focused on when defining Akoma Ntoso, while ensuring adequate flexibility to permit the nuances of each jurisdiction to be handled.

What's the best way to adopt Akoma Ntoso?

There are many approaches that can be successful, but in all cases it's best to adopt Akoma Ntoso incrementally rather than try to do everything at once. (This actually applies to many technologies, not just Akoma Ntoso.)

The first step is to learn how to model your local document traditions within Akoma Ntoso. If an existing XML rendition is available, this is easily done by building an XSL transform and some CSS files. Alternatively, you may have to resort to building a more comprehensive converter. In either case, be prepared to experiment with alternate ways to best represent your document structure.

Who has already adopted Akoma Ntoso?

There are already several existing implementations of Akoma Ntoso and a number of implementations in development. The jurisdictions that have already made a commitment to Akoma Ntoso include the European Parliament, the European Commission, the UK Parliament, the EU Publication Office, the Scottish Parliament, the Italian Senate, the Parliament of Uruguay, the Library of Congress of Chile, and several agencies within the United Nations including the Food and Agriculture Organisation (FAO).

Why does Akoma Ntoso seem so large and complex?

As a standard, Akoma Ntoso has a broad scope and is attempting to drive to some common practices. The more consistency we can realize among different

jurisdictions, the more know-how, tools, and experiences can be shared.

However, Akoma Ntoso realizes that excessive rules could hinder adoption. For that reason, a set of compliance levels have been established to provide guidance for how best to implement Akoma Ntoso.

You will, no doubt, find that you have to make a trade-off between being true to the Akoma Ntoso vision and being compatible with your own existing environment. That is to be expected and Akoma Ntoso provides the latitude that will allow you to choose how best to proceed, favouring your internal requirements with the *desirements* of Akoma Ntoso.

The next chapter will discuss the compliance levels and how you might choose which path to take.

Akoma Ntoso Pocket Guide

Chapter 2: Compliance Levels

Early on it was obvious that for Akoma Ntoso to thrive, there had to be enough flexibility in the specification to allow for an incremental adoption of the standard.

In order to facilitate this incremental adoption process, two dimensions of compliance have been defined — *Referencing* and *Metadata*.

For *Referencing*, some jurisdictions have a substantial investment in alternate referencing technologies such as ELI (European Legislative Identifier) that would be an alternative to Akoma Ntoso's own in-house referencing scheme.

Akoma Ntoso accepts two approaches for referencing compliance — first, using the in-house recommendation or alternatively, using a *functionally equivalent naming convention* (FENC). As long as the FENC supports certain characteristics of the in-house recommendation, it can be used instead. These characteristics include:

- *Meaningfulness:* the name is a meaningful and logical description of the resource.
- *Permanence:* the name is permanent and stable over time.
- *Invariance:* the name derives from invariant properties of the resource so as to provide certainty of deriving the same name.

The *Metadata* levels of compliance are:

- *Sublevel A:* Provide basic metadata — provide metadata fields necessary to identify the document, as prescribed.
- *Sublevel B:* Use the normative references of Akoma Ntoso — basically follow the referencing guidelines of Akoma Ntoso.
- *Sublevel C:* Use advanced metadata — use the enhanced metadata fields that more completely describe the legal effect of the document in a way that supports automation.
- *Sublevel D:* Use semantic elements — use the mechanisms Akoma Ntoso provides to weave information relating to an ontology you define into your document. Basically, this means tagging people, places, things, etc. in the text.

Don't worry if not all of this makes sense right now. We will go over much of this later on.

The next chapter will start to dig into the Akoma Ntoso tag-set by covering the different types of documents that Akoma Ntoso supports.

Akoma Ntoso Pocket Guide

Chapter 3: Document Types

While Akoma Ntoso has over 300 elements, you will find that many of them are simply refinements of a core structure that is reused over and over. The first place we are going to see this is in the document type. In Akoma Ntoso, there are six basic document structures. However, there are twelve actual document types. This is because some of the document types share their document structure with other document types.

Hierarchical structure — this is the document structure for documents that have a well-defined and regular document hierarchy. There are two hierarchical document types:

`<bill>` — a proposal for law, regulations, or other normative text.

`<act>` — an enacted law, regulation, or other normative text.

Amendment structure — this is the document structure for documents that amend other documents. There is a single amendment document type:

`<amendment>` — a proposed or adopted amendment.

Note: The notion of what is an amendment can vary form one jurisdiction to another. In some cases an amendment is a single change which can be grouped into an amendment list while in others a group of changes constitutes what is known as an amendment. Both cases can be modelled in Akoma Ntoso using the `<amendment>` and `<amendmentList>` (described below) document types.

Collection structure — this is the document structure primarily used for documents that are a composition of other documents. Thankfully, it is also possible to *inline* some of these composite documents for those cases, as described above, where they aren't really separate documents. There are three collection document types:

`<amendmentList>` — a collection of amendment documents.

`<documentCollection>` — a more generic composition of other documents.

`<officialGazette>` — a specific composition of documents that forms the official publication of the legislative body.

Debate structure — this is a document structure that models a recorded debate in the chamber or committee hearing. There is a single document type for this:

`<debate>` — the transcript of a chamber or committee debate.

Judgment structure — this is a document structure used to record a judge's opinion. There is a single document type for this:

`<judgment>` — a judge's opinion or ruling on a matter.

O*pen structure* — this is a catch-all structure that is very flexible and is intended to be used to model documents that are used in the legislative process and don't have the same level of formality or uniformity found in the other document types. There are three open document types:

`<doc>` — a generic document type intended to be used whenever another more suitable document type does not exist. One possible application of the <doc>

element is to contain non-positive titles of the U.S. Code. In this case, these are documents of law that are neither proposed nor enacted law, so using the <bill> or <act> tag would be erroneous.

<debateReport> — a summary or report about a debate (unlike a <debate> which is a transcription of what was said and done.)

<statement> — A statement from a chamber or other legislative body that either has no or has limited legal effect. An example of a statement in the United States would be a resolution (a word that seems to be quite overloaded with meaning around the world)

There are two attributes associated with the document type elements that are important:

@name — use this required element to either subtype the document type or to apply a more jurisdiction-specific term to the document type. For instance, some jurisdictions may refer to their acts as ordinances. You can use the @name attribute to specify that the document using the <act> element should be referred to as an *ordinance*.

@contains — use this element to declare that this document either contains the original version, a subsequent version, or contains all versions merged

together. Typically, a bill will contain a single version while an act (modelling a statute or code) may contain multiple versions reflecting the consolidation of amendments over time. If you don't use this attribute, it is assumed to contain the original version.

So, now we have covered the first twelve elements in Akoma Ntoso. The next chapter will show how these twelve elements are used in an Akoma Ntoso document — they're not the root element of the XML document as you might expect. We will cover the basic top level document structure that is found in all twelve document types.

Akoma Ntoso Pocket Guide

Chapter 4: Basic Document Model

All twelve Akoma Ntoso document types share a common root element. This element is `<akomaNtoso>`. This is similar to HTML which has the `<html>` root element, XSLT which uses the `<xslt>` root element, and XSD which uses the `<xsd>` root element. The root element defines the content model for the document that is contained within it — in this case declaring the document to be Akoma Ntoso.

Namespaces

After working in XML for many years, I've discovered that nothing causes more confusion than namespaces. Simply put, namespaces exist to prevent name collisions from occurring when you use elements from different models or schemas together. Namespaces are analogous to country-code prefixes on telephone numbers.

Typically, a namespace will be described by two pieces of information — the namespace URI(/IRI) and an optional prefix. The prefix is a short string of text

that will precede the element name unless the namespace is declared as the default.

On the other hand, the namespace URI is typically specified as a URL — and this leads to all sorts of confusion. URLs are usually intended to be pointers to a location on the internet. However, when used as a Namespace URI, they are merely being used to ensure a unique name to the schema. You may find the schema at the URL specified, but this is a convenience rather than a requirement. Don't expect to find the Akoma Ntoso schema at the Namespace URI for Akoma Ntoso — although it does resolve nicely to the OASIS information page relating to Akoma Ntoso.

For Akoma Ntoso, the namespace URI is currently:

```
http://docs.oasis-open.org/legaldocml/ns/akn/3.0/WD17
```

This reflects its current state as a working draft, with version 17 being the most recent version.

When Akoma Ntoso is ratified, the official namespace will be:

```
http://docs.oasis-open.org/legaldocml/ns/akn/3.0
```

If you're confused why this is version 3.0 of Akoma Ntoso, it is because two earlier versions predated the

OASIS standardization effort. The namespace URI reflects this historical artefact.

Typically, Akoma Ntoso is the default namespace; a prefix is not required. However, when a prefix is desired, the preferred prefix is `akn`:

Schema or DTD

The Akoma Ntoso standard is provided with only an XML Schema (XSD). While a DTD was available with earlier versions of Akoma Ntoso, the DTD is now deprecated.

Document Structure

With the root element and namespaces out of the way, let's now take a look at the overall document structure:

```
<akomaNtoso xmlns="http://docs.oasis-
open.org/legaldocml/ns/akn/3.0">
  <{documentType}>
    <meta>...</meta>
    <coverPage>...</coverPage>
    <preface>...</preface>
    <preamble>...</preamble>
    <header>...</header>
    <{bodyElement}>...</{bodyElement}>
    <conclusions>...</conclusion>
    <attachments>...</attachments>
    <components>...</components>
  </{documentType}>
</akomaNtoso>
```

Let's take a look at each element in this simple structure:

`<{documentType}>` — this is one of the twelve document type elements. In general, all twelve document type have a similar content model at the next level down, with some minor variation.

`<meta>` – this contains the primary metadata for the document. All documents are required to have this element, although the amount of content within it might vary considerably.

<coverPage> — (Optional) This element can contain any cover page to be associated with the document. The structure within it is very general. Depending on your internal practices, you might use this element or you might model the cover page as a separate document altogether.

<preface> — (Optional) This element contains any matter found at the beginning of a document. This often includes the document name, number, authors or sponsors, and the long and short title. As with the cover page, the content structure of the preface is very flexible. Typically, you will create a series of simple paragraphs using the <p> or <block> elements and then tag the various parts within these paragraphs using various inline elements that Akoma Ntoso provides.

<preamble> — (Optional) This element is used for some document types. In some jurisdictions, a bill and/or an act is required to have an enacting formula (or clause) which declares the source from which the law claims to derive its authority. Without it, it may simply not have the force of law behind it. Akoma Ntoso provides a <formula> element for this purpose.

Sometimes, especially in jurisdiction deriving from European legal traditions, you might also find that the enacting formula is preceded and/or followed by recitals. Recitals may indicate the legal basis, preparatory document, or state an issue or grievance that this document addresses. Recitals are specified using the `<recitals>` container with individual `<recital>` elements within it.

`<{body}>` — This is the main part of any Akoma Ntoso document. There are six different body elements, corresponding to the six document structures introduced in the preceding chapter:

`<body>` — This element models the hierarchical structure found in a `<bill>` or `<act>`.

`<amendmentBody>` — This element contains the structures needed to describe a single change.

`<collectionBody>` — This element allows a collection of documents to be composed. The documents can either be inline embedded or included by reference.

`<debateBody>` — This element contains the structures necessary to record a debate.

<header> — This element is used for the preface to a judgment.

<judgmentBody> — This element contains the structures necessary to describe a judge's opinion or ruling.

<mainBody> — This element contains a general set of structures which can be used in a variety of ways.

<conclusions> -- (Optional) This element is intended to contain all concluding matter including dates and signatures.

<attachments> -- (Optional) This element is intended to be used to either contain or reference parts of a document such as an annexes, appendices, related bills, or treaties that are not treated as part of the document.

<components> -- (Optional) This element is intended to contain or reference other independent documents that compose this document, but which are not considered as attachments. These documents might be schedules or tables which are appended to the main document. This structure is also used by document collections described in Chapter 21: Collections.

For reference, the table below shows which elements can be used with which document structures and the order of the elements:

	Hierarchical	Amendment	Collection	Debate	Judgment	Open
<meta>	Req.	Req.	Req.	Req.	Req.	Req.
<coverPage>	Opt.	Opt.	Opt.	Opt.	Opt.	Opt.
<preface>	Opt.	Opt.	Opt.	Opt.	-	Opt.
<preamble>	Opt.	-	Opt.	-	-	Opt.
<header>	-	-	-	-	Req.	-
<body>	Req.	-	-	-	-	-
<amendmentBody>	-	Req.	-	-	-	-
<collectionBody>	-	-	Req.	-	-	-
<debateBody>	-	-	-	Req.	-	-
<judgmentBody>	-	-	-	-	Req.	-
<mainBody>	-	-	-	-	-	Req.

In this chapter, we covered a lot of ground, laying out the basic structure of an Akoma Ntoso document. The next few chapters will look at the various parts that were outlined in this chapter, starting with the metadata element.

Akoma Ntoso Pocket Guide

Chapter 5: Understanding Metadata

Akoma Ntoso's metadata model is probably the most daunting part of Akoma Ntoso. In order to cut through this complexity, we're going to take the subject a bit slow, breaking it down in stages. For now, we will just look at the basic areas of metadata. Later on, we will delve into the details of each section.

The `<meta>` element contains up to nine different container elements in the order shown below, each one describing a different aspect of the document:

`<identification>` — This is always the first element and it is the only required part of the metadata. It describes how the document is identified using something called FRBR (Functional Requirements for Bibliographical Records). We will dive into this in much more detail later but for now all you need to know is that the identification section contains a number of URIs which can be used to describe the various aspects of this document.

<publication> — (Optional) This element, at the top level of the metadata, describes the official publication event for the document.

<classification> — (Optional) This element is used to create a list of keywords to be associated with this document.

<lifecycle> — (Optional) This element is used to describe the set of events that resulted in the current state of the document. This is usually a granular enumeration of the initially created document and then the updates that subsequently amended it.

<workflow> — (Optional) This element is similar to the lifecycle, but it describes the formal stages that the document has been through. For instance, a bill may be introduced, go through various committees and/or readings, be passed, and then finally enacted into law. The <workflow> container describes this history.

<analysis> — (Optional) This element contains various elements that reflect an analysis of the document and which describe what the effect of the document is to be, In other words, this section describes the modifications this document contains or is making to other documents.

<temporalData> — (Optional) This element is used for one of the most advanced capabilities supported by Akoma Ntoso — the ability to describe the entire historical text of a document, associating different text with different time periods. The <temporalData> container is used to define time periods which are used when associating text with the time period for which it applies.

<references> — (Optional) This element is used to contain all the references contained within the document. Many of these references will be ontological references to Top Level Classes defined in an ontology. This is an advanced feature which we will come back to later.

<notes> — (Optional) This element is used to collect editorial notes relating to this document or to items within the document.

<proprietary> — (Optional) This is another open element intended to contain anything that is specific to your particular environment. Often this will include elements required by tools, legacy data that has no other location within an Akoma Ntoso document, or classification data that goes beyond what Akoma Ntoso requires.

`<presentation>` — (Optional) This is an open element intended to contain anything related to the presentation or publication of the document. Akoma Ntoso does not prescribe the contents of this element — it is for you to use as necessary. You will most likely want to use your own element names in your own namespace.

There is one attribute, at this point, that you need to be concerned with. All the elements found at the top level of the `<meta>` container element, except for the `<publication>` element, require the `@source` attribute. This attribute contains the identification of the person or organisation that provided that metadata. The correct way to use this attribute is a little involved and requires getting into some aspects of the metadata that we have not yet covered. Rather than dive into this now, we will leave the details for later and just show the basic technique:

```
<meta>
  <identification source="#legCounsel">
   ...
  </identification>
  <publication .../>
  <references source="#legCounsel">
    <TLCOrganization
     eId="legCounsel"
     href="/akn/cc/organization/legcounsel"
     showAs="Legislative Counsel"/>
  </references>
</meta>
```

Note: In general, the OASIS Technical Committee has chosen to stick to traditional English spelling rather than American spelling for element names and for the language of the specification — LegalDocML being an international effort. The element `<TLCOrganization>` is one example where the American spelling has, nonetheless, prevailed.

Of the elements described above, only the <identification> element is required. The publication element can occur, at most, one time. All other elements are optional and can be repeated, as necessary. The reason you may want to have more than one instance of any of these has to do with the `@source` attribute. If the metadata comes from more

than one source, you would want to have a separate container element for each source.

As you can tell, the metadata in Akoma Ntoso is quite involved and can take quite an effort to understand and implement. However, much of this can be relegated to advanced features which can be learned only when and if needed. What's most important is that you understand that these features are there when you need them.

The next chapter will finally get to the heart of a document — the document hierarchy. This is the core structure of most Akoma Ntoso documents.

Akoma Ntoso Pocket Guide

Chapter 6: Document Hierarchy

The primary structure of Akoma Ntoso documents that follow the hierarchical structure is a hierarchical level model contained within the `<body>` element.

In many jurisdictions, the hierarchical levels can be divided into two different types of levels — upper levels and lower levels. Usually, it's the section or article that separates the two types of levels. In Akoma Ntoso, there are a number of elements provided to model all these levels. The following list defines these level names:

`<title>` — not to be confused with the `<docTitle>`, `<longTitle>`, or `<shortTitle>` in the preface.

`<subtitle>`

`<division>`

`<subdivision>`

`<part>`

`<subpart>`

<chapter>

<subchapter>

<article>

<section>

<subsection>

<paragraph>

<subparagraph>

<clause>

<subclause>

<list>

<sublist>

<rule> — often a subclassing of a section

<subrule>

<level>

Don't worry if you don't see a specific level you use in this list. Next, we will discuss how to synthesize levels using the generic elements provided by Akoma Ntoso.

Akoma Ntoso makes no assumption about the hierarchical organization of these levels. There is considerable variation in how these levels are

arranged, sometimes even within a jurisdiction's own documents. Also, there is some variation in the actual meaning of the terms. For instance, in many U.S. jurisdictions, an Article is a container of sections. You see this in the U.S. Constitution which has seven articles, each containing several sections.

However, in European legal traditions, the role of the section and article are reversed — being more similar to a newspaper's notion that a section is made up of articles. In fact, HTML5 follows this model as well. In U.K. tradition, in some types of documents, an article is a type (or subclass) of a section. For Akoma Ntoso, it doesn't matter. All notions of an article and section can be accommodated.

Another special situation you might encounter is a case where the hierarchical levels are either not named or their naming is unclear. For instance, in some legal traditions that are less influenced by European or U.K. legal traditions, the upper levels are denoted by a number and location in the hierarchy. Another similar situation is found in U.S. legal traditions when the lower levels are cited something like *§5(a)(2)* where it is not clear what the lower levels are named (and often even the terminology found in the documents is inconsistent). For these undefined

or inconsistent cases, the more generic <level> element may be used.

Note: The U.K. uses a citation notation that is similar to the one shown above, but the levels names tend to be more uniformly understood.

In some legal traditions, you might come across an unnumbered heading that acts as a divider more than a hierarchical container. Usually, this heading is shown centred. This is called a *crossheading* and can be modelled using the <crossheading> element. It does not participate in the hierarchy. In some cases, it might be desirable to create an artificial hierarchy for cross headings. This can be done as follows:

```
<level>
  <crossheading>...</crossheading>
  ... other levels ...
</level>
```

In addition to the basic hierarchy described above, there is a collection of other similar elements which may be useful, depending on your needs. These include the <book>, <tome>, <point>, <indent>, <alinea>, <proviso>, and <transitional> elements.

The next chapter will cover the details of a level itself, showing the different ways to configure a level.

Akoma Ntoso Pocket Guide

Chapter 7: Level Content

There are basically two different ways to configure a level — either as text content, or with sublevels. While the structures for these two cases are quite different, both structures share the same initial elements.

The heading of a level can be made up of a number of elements in a variety of configurations. There are three elements that are important:

`<num>` — (Optional) This element contains the alphanumeric designation assigned to the level. There is a best practice that needs mentioning when referring to the `<num>` element. There is a natural instinct to leave any type designation off of the numeric designation. For instance, showing the num as `<part><num>1</num>...</part>` rather than `<part><num>Part 1</num>...</part>`. We call text that is added by rule *generated text*. It is a very problematic practice. There are many reasons for this

including that it makes amending techniques far more difficult than they need to be and that it requires special knowledge for generating the text – that might not exist when a document is transmitted to an external party. Instead, we always strive to ensure that every single word that is shown in the printed form actually exists, in the correct sequence, in the XML document as general content.

`<heading>` — (Optional) This element contains a primary heading.

`<subheading>` — (Optional) This element contains a secondary heading and should only be used in conjunction with a primary heading.

All three of these elements are optional and the order is unspecified. This is to support the variety of headings found around the world. Some examples of heading configurations you might encounter include:

- **<num> <heading>**
- **<num> <heading>**
  <subheading>
- **<num>** - <content>

- **<heading>**
  <num> - <content>

- **<heading>**
  <num> - *<subheading>* (be careful not to mistake a cross heading with a heading. A cross heading would usually apply to more than a single subsequent item)

Akoma Ntoso is designed to support any of these configurations as well as others you may need.

After the initial elements come the two basic configurations:

The simplest levels merely contain text. This text must be enclosed within a single <content> element designed to contain basic text structures. These text structures mostly mimic equivalent structures found in HTML. For example, Akoma Ntoso offers basic elements such as <p>, <ul>, <ol>, and <table>. We will delve into these elements later on. In addition, Akoma Ntoso allows a few other elements in the <content> element. Use the <foreign> element to wrap around a foreign construct from other namespaces such as MathML for mathematical formulas and ChemML for chemical formulas. Finally, there are some generic elements

that are permitted, but we will discuss those in more depth later on when we discuss generic elements.

For hierarchical levels that contain sublevels, the structure is bit more complex. The basic model is:

```
<{levelName}>
   <{num}> <{heading}>
  [<intro>]
   <{subLevelName}>...</{subLevelName}>
   ...
  [<wrapUp>]
</{levelName}>
```

where the <intro> and <wrapUp> are optional and follow the same content model as the <content> element. Most often, the <intro> is used to contain some introductory text followed by either a ':' or an '—' (em-dash). This fragment of text goes by many names — a chapeau, opening words, or lead-in text among others.

In some very large documents, often codes, certain levels will contain their own table of contents. Use the <toc> element within the <intro> to define a table of contents.

The <wrapUp> element is used to include trailing text that often follows a series of level elements. Note that provisos, which often precede other closing text, have

their own `<proviso>` element. Provisos are identified as starting with *Provided that* or something similar to that effect.

There is one particular case that deserves special mention — numbered and unnumbered lists. Sometimes, numbered lists look much like any other hierarchical lower level, starting with a number and then followed by text. One hint that you're looking at a list is that the opening text, the list items, and the closing text are all written in the form of a single sentence, as follows:

> 1) The following items—
> (a) the first item;
> (b) the second item; and
> (c) the last item
> are written as a single sentence.

There are three different ways to handle this. The first approach is to use the <list> construct as part of the hierarchy:

```
<section>
  <num>Section 1</num>
    <list>
      <intro>
         The following items--
      </intro>
      <point><num>(a)</num>
         the first item;</point>
      <point><num>(b)</num>
         the second item; and<point>
      <point><num>(c)</num>
         the last item</point>
      <wrapUp>
         are written as a single
         sentence.
      </wrapUp>
    </list>
</section>
```

There are a few alternatives to this approach. First, the introductory text and/or the wrap up text could be written as part of the section rather the list – moving the <intro> and/or <wrapUp> elements to outside the <list> element. Second, an <indent> element can be used in place of the <point> element.

The second approach is to use the <blockList> construct as part of the content:

```
<section>
  <num>Section 1</num> <content>
    <blockList>
      <listIntroduction>
         The following items--
      </listIntroduction>
      <item><num>(a)</num>
         the first item;</item>
      <item><num>(b)</num>
         the second item; and<item>
      <item><num>(c)</num>
         the last item</item>
      <listWrapUp>
         are written as a single
         sentence.
      </listWrapUp>
    </blockList>
  </content>
</section>
```

With this approach, the introductory and wrap up text can only be treated as part of the list, not the surrounding section.

Importantly, with both the first and second approaches, all text is explicitly shown – no text is generated when there is a numeric designation or a bullet to show.

Alternately, the third approach allows you to treat the list as normal text within a <content> element and use the HTML inspired ordered list as follows:

```
<section>
  <num>Section 1</num> <content>
    <p>The following items--</p>
    <ol>
      <li>the first item;</li>
      <li>the second item; and<li>
      <li>the last item</li>
    </ol>
    <p>are written as a single
       sentence.</p>
  </content>
</section>
```

If you know HTML, this should look familiar. There are drawbacks to this approach, however. First, there is no particularly good method to indicate introductory and/or wrap up text. Secondly, the numbering of the list items or the bullet items are generated text. As already noted, generated text is very problematic for things like amending and we try to avoid that.

The next chapter will scratch an itch that cannot be put off any longer — how do you handle concepts that aren't already built into Akoma Ntoso. We will look at the generic model and the types of options it provides.

Akoma Ntoso Pocket Guide

Chapter 8: The Generic Model

Akoma Ntoso is actually based on a relatively simple foundation. Every Akoma Ntoso element is built by refining one of seven basic building blocks. Each building block defines a specific content structure.

In addition to all the specific tags that are provided, there are also tags for each of the several building blocks along with an attribute, `@name`, which is used to name your custom element. These building blocks are *generic* elements and can be used in many of the places where the specific elements can be used.

The generic elements are:

`<hcontainer>` - This is used to define a hierarchical container — generally a level in the document hierarchy. An `<hcontainer>` contains normal hierarchical structures such as `<num>`, `<heading>`, `<subheading>`, `<intro>`, other levels, and `<wrapUp>`.

`<container>` - This is used to define a container element that only contains other elements – and not

text. A <container> can be placed in the <preface> and <preamble> among other places.

<tblock> - This is used to define title blocks. Title blocks are block level structures with heading, subheadings, and/or numeric designations.

<block> - This is used to define a container that can contain text along with inline elements.

<inline> - This is an element that wraps around text and/or elements without causing a block structure or presentation — similar to a element in HTML.

<subFlow> - This is used to define a block level or hierarchical structure within an inline structure. This often occurs when embedding text into a document.

<marker> - This is an element that defines a point in the document — and has no content of its own.

As previously mentioned, you use the @name attribute to name your element. For instance, to create a *schedule* element that behaves as a hierarchical level, you would use:

```
<hcontainer name="schedule">
 ...
</hcontainer>
```

Occasionally, you may find that a generic element isn't available where you want one or you may wish to further refine an existing element. There is a way to do this, and you may already be familiar with the concept. Akoma borrows two attributes (and others) from HTML. These are the `@class` and `@style` attributes. Much as you might use the `@class` attribute in HTML to refine a specific HTML element, you can use the `@class` attribute in Akoma Ntoso to refine an Akoma Ntoso element. For example, you might wish to refine the `<blockList>` element to specify that it is a numbered list. You could do this as follows:

```
<blockList class="numberedList">
 ...
</blockList>
```

By now it should have become apparent that Akoma Ntoso borrows a lot from HTML. The next chapter will focus on how Akoma Ntoso builds on many aspects of HTML.

Akoma Ntoso Pocket Guide

Chapter 9: Using HTML

Akoma Ntoso borrows many elements and attributes from HTML. These are used in many cases where you are trying to model specific constructs needed in a document – but which either don't have or don't need the semantic richness of other Akoma Ntoso elements.

The elements borrowed from HTML include:

`<p>` – Be careful not to confuse the use of this element with a formal numbered paragraph.

`<span>` – Consider using `<inline @name='{name}'>` instead.

`<br>`

`<b>`

`<i>`

`<u>`

`<sup>`

`<sub>`

<abbr>

<a> — Use this for references to non-legal texts. Otherwise, use <ref> or <rref>.

 — Consider using <blockList> instead.

 — Consider using <blockList> instead.

<table>

<caption>

<tr>

<th>

<td>

Note: Akoma Ntoso does not include HTML's <thead> and <tfoot> elements for tables. That's no big loss.

When it comes to attributes, Akoma Ntoso borrows six attributes from HTML:

@class — This attribute is used to specify a CSS class for presentation.

`@style` — This attribute is used to specify CSS attributes inline.

`@title` — This attribute is used to specify additional information about an element, and is usually shown in a tooltip.

`@href` — This attribute is used to refer to another document or provision.

`@src` — This attribute is used when including or incorporating a resource contained in another file.

`@alt` — This attribute provides text to display in case the referenced resource cannot be retrieved.

Note: While both Akoma Ntoso and HTML share the `@name` attribute, the usage is different.

So far we have learned about many Akoma Ntoso elements and attributes include a few that are borrowed from HTML. The result is an extensive set of elements and attributes that can model many constructs found in legislation. However, there are still gaps. The next chapter will describe how to create a composite document which includes foreign namespaces.

Akoma Ntoso Pocket Guide

Chapter 10: Using Other Namespaces

While Akoma Ntoso provides an extensive set of elements (over 300) and a great many attributes, there are nonetheless many gaps that either were not anticipated or that are too specialized to be included within Akoma Ntoso. Examples where this occurs is in mathematical formulas and chemical formulas. In both of these cases, well-established namespaces exist and it would be counter-productive to attempt to incorporate all those elements and attributes within Akoma Ntoso. For one thing, many tools exist which understand these namespaces and they would not understand Akoma Ntoso's equivalent elements and attributes.

Akoma Ntoso addresses this need in three specific ways:

1. First, Akoma Ntoso permits custom attributes to be added to any element. To

do this, you must first declare your additional namespace prior to using a custom attribute and it must start with an appropriate prefix.

2. Second, content elements such as `<content>`, `<intro>`, and `<wrapup>` permit the use of the `<foreign>` tag which is intended to contain elements of another namespace as follows:

```
<content>
  <p>...</p>
  <foreign>
    <math xmlns="…/MathML">
      <mrow>
        <mi>a</mi>
        <mo>&InvisibleTimes;</mo>
        <msup>
          <mi>x</mi>
          <mn>2</mn>
        </msup>
        <mo>+</mo>
        <mi>b</mi>
        <mo>&InvisibleTimes;</mo>
        <mi>x</mi>
        <mo>+</mo>
        <mi>c</mi>
      </mrow>
    </math>
  </foreign>
  <p>...</p>
</content>
```

As is normal practice, you can either redefine the default namespace as shown, or use a namespace prefix for your foreign namespace. For example, you could use the prefix `m` for mathml as in `<m:math xmlns:m="…/MathML">`.

3. The third way Akoma Ntoso allows non-Akoma Ntoso content has already been

covered. In the <meta> container, the <proprietary> tag allows foreign metadata to be recorded.

The next chapter will change direction and return to the metadata, focusing on the FRBR (pronounced Ferber) model for identification. This will lay the foundation for much of the upcoming discussion of the metadata model.

Akoma Ntoso Pocket Guide

Chapter 11: Identifying a document using FRBR

The Functional Requirements for Bibliographical Records (or FRBR and pronounced as Ferber) provides a fundamental model for referring to and identifying documents in Akoma Ntoso.

You can learn more about FRBR at http://www.ifla.org/publications/functional-requirements-for-bibliographic-records.

This is an abstract model defined by librarians for categorising books in a library. While FRBR is a subject unto itself, there are only a few core concepts that you really need to understand. Essentially, FRBR defines four basic concepts which are important to Akoma Ntoso:

1. *The work* — this is the basic document being identified. Think of it as the name of a book such as *War and Peace* by Tolstoy.
2. *The expression* — this is the version or variant of the book. There might be many versions of

a book — such as editions or translations into other languages,

3. *The manifestation* — this is the manner in which it has been rendered. For instance, a book might come as hard or soft cover or even as an eBook.
4. *The item* — this is a particular instance of the book. For instance, the library downtown might have a copy and so too might a nearby bookstore.

In Akoma Ntoso, we mostly deal with the work, the expression, and the manifestation. However, all four cases do arise, although the item has less formality in Akoma Ntoso. For instance, a bill may be available from different sources. For instance, Cornell's LII might provide a copy, the Library of Congress might provide a copy, the U.S. Congress themselves might provide a copy, and a legal publisher such as Lexis Nexis might provide a copy. Each may have a separate *instance* of the same document.

In Akoma Ntoso, the `<identification>` portion of the metadata is used to provide a set of URIs that can be used to retrieve a document, specifying it at the various FRBR levels.

There are four elements in the <identification> portion of the metadata, corresponding to the four levels of FRBR:

<FRBRWork>

<FRBRExpression>

<FRBRManifestation>

<FRBRItem>

Only the first three elements are required, in the order shown above.

Contained within each FRBR* container is a series of elements that describe various aspects of the FRBR level. Which elements are required or optional for each level is summarised below:

	Work	Expression	Manifestation	Item
<FRBRthis>	Req.	Req.	Req.	Req.
• Provides the URI to refer to this specific file at each of the four levels.				
<FRBRuri> *	Req.	Req.	Req.	Req.
• Provides the URI to refer to this specific document at each of the four levels				
<FRBRalias> *	Opt.	Opt.	Opt.	Opt.

	Work	Expression	Manifestation	Item
• Allow you to associate alternate names or URIs to this document				
`<FRBRdate> *`	Req.	Req.	Req.	Req.
• Provides for dates for this document				
`<FRBRauthor> *`	Req.	Req.	Req.	Req.
• Provides for authors (or contributors) for this document				
`<FRBRcountry>`	Req.			
• Allows you to associate a country (or jurisdiction) with this document				
`<FRBRsubtype>`	Opt.	-	-	-
• Allows you to subtype this document (related to the @name attribute on the doc type element)				
`<FRBRnumber> *`	Opt.	-	-	-
• Allows you to associate a number with this document				
`<FRBRname> *`	Opt.	-	-	-
• Allows you to associate a name with this document				
`<FRBRprescriptive>`	Opt.	-	-	-
• Allows you to declare whether or not this document contains prescriptive text (i.e. text for laws or regulation)				
`<FRBRauthoritative>`	Opt.	Opt.	-	-
• Allows you to declare whether or not this text is authoritative (i.e. official)				
`<FRBRmasterExpression>`	-	Opt.	-	-
• Allows you to declare that this expression of the document is the master — which is used in conjunction with the @wId attribute which we have yet to cover.				
`<FRBRversionNumber>`	-	Opt.	-	-

	Work	Expression	Manifestation	Item
• Allows you to associate a version number with this document.				
`<FRBRlanguage>`	-	Req.	-	-
• Allows you to declare the language of this document				
`<FRBRtranslation>`	-	Opt.	-	-
• Allows you to state that this document is a translation from another text.				
`<FRBRportion>`	-	-	Opt.	-
• Allows you to associate a document portion with its origin.				
`<FRBRformat>`	-	-	Opt.	-
• Allows you to associate an Internet Media Type specification to the document.				

** Items marked with an asterisk are permitted multiple times*

There are many attributes used by these various elements:

`<FRBRthis>, <FRBRuri>, <FRBRalias>, <FRBRcountry>, <FRBRsubtype>, <FRBRnumber>, <FRBRprescriptive>, <FRBRauthoratative>, <FRBRmasterExpression>, <FRBRversionNumber>, <FRBRformat>`

`@value` — Specifies a general value with the element.

`@refersTo` — (Optional) Connects this element to an item in the ontology.

`@showAs` — (Optional) Provides a textual way to describe this element.

`@shortForm` — (Optional) Provides an abbreviated form of the @showAs value.

`<FRBRdate>`

`@date` — Associates a date with the document using the XML Schema syntax of YYYY-MM-DD or, when more precision is required YYYY-MM-DDThh:mm:ss(zzzz).

`@name` — Gives the date a name to be known as (i.e. enacted).

`<FRBRauthor>`

`@href` — Usually contains an id reference (ie. `#ref`) to a person or organisation defined in the ontology.

`@as` — (Optional) Specifies the role of the author (i.e. sponsor, drafter, editor) as an id reference to a role defined in the ontology.

`<FRBRlanguage>`

`@language` — Specifies the language of this document using a code from RFC 4646.

`<FRBRtranslation>`

`@href` — Provides a reference to the source document that this translation came from.

`@fromLanguage` — Specifies the language translating from, selecting a code from RFC 4646.

`@by` — Specifies the translator using an id reference (i.e. `#ref`) to a person or organisation defined in the ontology.

`@authoritative` — (Optional) Specifies, as a Boolean `true` or `false`, if this translation is authoritative.

`@pivot` – (Optional) Specifies an intermediate human language through which the translation was made. The value is a code from RFC 4646

`<FRBRportion>`

`@from` — Specifies an id reference to the portion of the full document that has been extracted. If the portion contains a range of

elements, then `@from` specified the first id reference.

`@upTo` — (Optional) Specifies, when the portion contains a range of element, the last id reference.

`@refersTo` — (Optional) Connects this portion to an item in the ontology.

`@showAs` — (Optional) Provides a textual way to describe this portion.

`@shortForm` — (Optional) Provides an abbreviated form of the `@showAs` value.

There are two more important elements found in the `<identification>` portion of the metadata:

`<componentInfo>` — Contains a list of files which collectively form this document. For example, the main text might be found in a document named `main.xml` and any associated attachments in secondary files such as `annex.xml` or `schedule1.xml`. Each file is described in the `<componentData>` element as follows:

```
<componentData
  eId="..."
  href="{fileName}.xml"
  name="{name}"
  showAs="{visibleName}"/>
```

The `@eId` attribute will be introduced later.

`<preservation>` — Provides a means for recording any preservation action which might have been taken to preserve this document. Akoma Ntoso does not prescribe what this element contains — you are free to use it as you need, using elements from your own namespace.

Now that we have cut through the complexity of the `<identification>` container, we need to understand how the various URIs found in this container are formed. The next chapter will dive into the Akoma Ntoso URL-based referencing scheme and learn more about a *Functionally Equivalent Naming Convention* (FENC).

Akoma Ntoso Pocket Guide

Chapter 12: URL-Based References

Akoma Ntoso provides an extensive URL-based scheme for identifying and referring to document resources in a legislative environment. This approach provides references suitable to interpretation by an HTTP-based system.

This reference scheme is part of the Akoma Ntoso Naming Convention and takes the following general form:

```
/akn/{jurisdiction}/{docType}/{docNumber}/[{lang}]@[{version}]/!{component}.{format}
|- Work ------------------------------|
|- Expression ------------------------------------------|
|- Manifestation ---------------------------------------------------------|
```

Where:

`/akn` — Declares the URL to be using the Akoma Ntoso referencing scheme.

`{jurisdiction}` — Declares the jurisdiction as a country code, possibly followed by a dash and any further specification that might be necessary. For example, the state of California would be `'us-ca'`.

`{docType}` — Declares the type of document.

`{docNumber}` — Associates a number with the document. In some cases, this includes multiple hierarchical levels.

`{lang}` — (Optional) Specifies the language of the document as an RFC 4646 language code.

`{version}` — (Optional) Specifies a version label (or date) for the expression.

`{component}` — Specifies the file component of the document.

`{format}` — Specifies an extension related to the file type being identified or requested (in a reference). For Akoma Ntoso documents, this should be `'xml'`.

Notes:

- The FRBRthis URI is usually the appropriate Work/Expression/Manifestation part PLUS the component part.
- The component part implies that a document is composed of multiple files, presumably grouped into a folder (or within an archive file). However, this isn't always the case. If a document does relate to a single file, the component is still referred to as `'!main'`.
- The FRBRuri URI is usually just the appropriate Work/Expression/Manifestation part
- Both the language and the version qualifiers are optional for an Expression, although usually at least one will be present. The @ sign will always be present when referring to an expression.
- An Item is usually specified using a full URI including the domain where the document can be found.

This referencing scheme has proven to be a bit controversial. The reason is that some jurisdictions already have a commitment to alternate referencing schemes that appear to offer equivalent functionality. Switching to Akoma Ntoso is sometimes politically difficult and/or will wreak unnecessary havoc upon

the rest of the (non-Akoma Ntoso-based) legislative infrastructure.

In order to mitigate this concern, it is possible to use an existing referencing scheme, referred to as a *Functionally Equivalent Naming Convention* (FENC), provided it truly offers an equivalent capability. One requirement, for documents that are to be published as Akoma Ntoso XML to the public at large, is that the `<FRBRalias>` be used to provide the Akoma Ntoso equivalent URL in order to foster interoperability among systems.

In order to make a complex URI scheme such as this work, there needs to be something that can interpret these URIs and map them appropriately into calls to the file system, a document repository, or an XML database.

Such a system, usually coupled with the web server, is called a *resolver*.

Before we can explore references, there is one more thing we need to cover — how provisions are identified. The next chapter will cover identifier theory and the various identifier attributes.

Akoma Ntoso Pocket Guide

Chapter 13: Identifiers

Akoma Ntoso offers an extensive identification mechanism — which supports two alternate use cases. Which you choose will depend on practical considerations that include tools you use and compatibility with existing conventions. Before getting to the two use cases, let's cover some fundamentals.

It is important to be able to identify each provision within a legal document uniquely such that a robust means of referring to that provision can be provided. In general, there are three important ways in which you will need to refer to and track a provision:

1. First, you will want to be able to track a provision throughout its life, knowing with certainty that you're referring to the same provision despite any moves or renumbering that it might experience. In essence, you need an *immutable* identification — a means of identifying a provision that will never change.
2. Second, you will want to be able to refer to each *version* of a provision, knowing with

certainty, that the text you have retrieved is exactly the text as it was when it was referred to.

3. Third, you will want to be able to refer to the *current* version of a provision as it changes over time, retrieving the latest text at the time it is accessed.

There are a few considerations which must be considered when developing an identification scheme:

- Can the identifiers be reliably produced? This is a special consideration when it comes to conversion. If your process relies on re-conversion of documents repeatedly, will each conversion yield the same identifiers?
- Can the identifiers be reliably managed? Special consideration must be given to how provisions are moved and copied. For instance, if a provision is moved from one location to another, its *immutable* identifier should be retained. However, if a provision is copied, then a new *immutable* identifier should be assigned — otherwise there will be multiple provisions with the same identity. A case which requires particular consideration is cut & paste. After a cut, the first paste

should be considered a move and any subsequent paste should be considered as a copy (this is not fool-proof, but it's the best approach possible without burdening the drafter with complex details).

- Can the identifiers be guaranteed unique? This consideration often comes down to the legislative practices and what is allowed or not. For instance, some jurisdictions permit and quite routinely have multiple provisions with the exact same number. This sometimes occurs erroneously but must be tolerated or it may be deliberate as alternate versions of a provision may have conditional logic that determines which is in force. Another consideration is that completely unrelated provisions may share a number — but over different periods of time. Quite often, ambiguities such as these are resolved in textual citations using *qualifying language* such as 'as introduced by ...' or 'as amended by ...'.

Akoma Ntoso provides three identifiers which can be used to address these needs:

`@eId` is the *expression* identifier. It is a user-meaningful token generated using a combination of

hierarchical location, document type, and document number as needed to ensure (some degree of) uniqueness. The `@eId` of a provision may change multiple times throughout the life of the document.

`@wId` is the *work* identifier. It uses the same syntactic convention as the `@eId` value. It is only assigned a value once the `@eId` changes, inheriting the original value, and will never change once assigned.

`@GUID` is an identifier assigned as a *globally unique identifier* (GUID) — a set of randomly generated alphanumeric values.

Now that we have covered the fundamentals, let's get back to the two use cases. Typically, you will use two of the three Akoma Ntoso identifier attributes. While there are many ways you might choose to use them, there are two specific uses cases for which they are intended:

1. Use `@eId` and `@wId` — In this usage, the manner in which the expression and work identifiers apply is a bit intertwined. For the vast majority of provisions that will only have a single location and numeric designation, there is no need to have more than a single identifier at all. The `@eId` value

will suffice as both the overall identifier (the *immutable* identifier) and the version identifier. However, as the `@eId` isn't immutable, it can change at some point. When this occurs, the `@wId` identifier comes into play, being introduced when necessary to hold the original `@eId` value once the `@eId` takes on a new value.

With this scheme, a provision is primarily known by its current address, XXX. Should it ever move or get a new address, it's then known as the provision at YYY that was originally found at XXX.

2. Use `@eId` and `@GUID` — In this usage, the `@eId` and `@GUID` have totally orthogonal roles. The @eId is the version identifier and describes the provision as it is located and numbered for this specific version. The `@GUID`, unlike the `@wId`, is assigned when the provision is first created and is maintained, unchanged, throughout the life of the provision.

   With this scheme, a provision is known in one of two ways. First, like the other scheme,

you can address by its current address as in XXX. However, if you wish to refer to the provision regardless of its current address and know that you're getting the same provision, then you can refer to it by the `@GUID` identifier it was issued at inception (`#1234`).

Neither scheme handles every conceivably issue and Akoma Ntoso, while attempting to provide a comprehensive solution for identifiers, does not provide guidance on every issue you may encounter. Both these schemes have issues which will need creative solutions. The `@eId/@wId` scheme has significant holes that that could result in naming collisions. The `@eId/@GUID` scheme reduces the size of these holes but does not eliminate them.

The second scheme is more likely (at least in my experience) to map into legacy identifier schemes, but it is difficult to ensure that the same identifiers are produced when repeatedly converting the same documents over and over if they don't already have GUID-based identifiers assigned from which to work from.

Now that we understand the motivation for the various identifiers, it's time to learn how to form an

identifier from a document's hierarchical location, its type, and its numeric designation. The next chapter will describe how to form an identifier in various scenarios.

Akoma Ntoso Pocket Guide

Chapter 14: Forming identifiers

Akoma Ntoso provides considerable guidance for how to form an `@eId` or `@wId` identifier. First, there is the overall structure of the identifier. Second, there is a set of required abbreviations to use for various provision types.

Format of an Identifier

The objective of an Akoma Ntoso identifier is to form a human-readable identifier that can easily be parsed and is, to a reasonable degree, guaranteed to be unique.

The @eId and @wId are expressed using a hierarchical notation based on the hierarchical level type and the alphanumeric designation assigned to it. The format is:

```
{levelType}_{levelNum}[__{sublevel}]
```

Where:

`{levelType}` — The level type is the name of the level, expressed in full if five or fewer characters or

by using abbreviations shown below. The level type is always expressed in lowercase. If a level does not have specific type designated, you can omit the `{levelType}_` part.

`{levelNum}` — The level number follows an underscore and is the exact numeric designation assigned to the level. The level number is expressed as a case-sensitive string with all modifiers and separators stated as is.

`[__{sublevel}]` — The sublevels are one or more hierarchical sublevels required to uniquely identify the provision. Each sublevel uses the same name pattern as the initial level — namely `{levelType}_{levelNum}`. Sublevels are preceded by double underscores.

With some levels, such as sections or articles, the numeric designation runs on throughout the document without resetting at each hierarchical boundary, and it is not customary to refer to the provision in a hierarchical manner. In this case, the identifier should only include the provision's own level type and level number.

Abbreviations

Akoma Ntoso provides the following set of preferred abbreviations to use as the level type:

XML element	Abbreviation
alinea	al
article	art
attachment	att
blockList	list
chapter	chp
citation	cit
citations	cits
clause	cl
component	cmp
componentRef	cref
documentRef	dref
fragment	frag
intro	intro
list	list
listIntroduction	intro
listWrap	wrap
paragraph	para

XML element	Abbreviation
quotedStructure	qstr
quotedText	qtext
recital	rec
recitals	recs
section	sec
subchapter	subchp
subclause	subcl
subparagraph	subpara
subsection	subsec
wrapUp	wrapup

For words greater than five characters in length that aren't on this list, choose a reasonable abbreviation as necessary. For instance, you might want to abbreviate 'schedule' as 'sched'.

Now that we understand how documents are identified using FRBR and referenced as URIs and how provisions are identified, it is time to learn how to make references between documents and provisions in documents. The next chapter will cover references.

Akoma Ntoso Pocket Guide

Chapter 15: References

Referring to documents or provisions.

Fundamentally, there are three types of references that can be made:

1. A reference to another document. This type of reference is called a document reference and is expressed as a URI to the document using the URL-based references of the Akoma Ntoso Naming Convention described earlier or a FENC reference described in Chapter 2: Compliance Levels.
2. A reference to a provision within the current document. This type of reference is called an internal id reference and is expressed as `#{eId}` where `{eId}` is the `@eId` attribute value of the provision being referenced.
3. A reference to a provision within another document. This type of reference is called an external id reference and can be expressed, using references that conform to the AKN naming convention, in two ways. The first

form, `{url}#{eId}` is a reference to the document containing the provision with a request to scroll to that element. The second form, `{url}~{eId}` is reference directly to the provision. The difference between the two forms is that the first form retrieves the whole document while the second form only retrieves the provision.

Referencing Elements and Attributes

Akoma Ntoso offers a few reference elements along with a set of attributes for establishing references.

There are four elements that can be used for references along with an entire portion of the metadata for references:

`<a>` — Similar to the HTML element with the same name, this element should only be used to make references to non-legal texts.

`<ref>` — Use this element to make a legal reference to a provision or to a document.

`<rref>` — Use this element to refer to a range of provisions rather than a single provision. If you're referring to a range of provisions within another document, you will need to wrap a <ref> around this element.

<mref> — Use this element to group a set of <ref> or <rref> elements together. The <mref> element is not an actual reference in itself. It's a grouping construct.

There are a few other referencing elements that have more specific uses:

<affectedDocument> — Use this element to refer to a document affected by this document.

<relatedDocument> — Use this element to refer to a document that this document is a report of.

<documentRef> — Use this element in a collection document (described in Chapter 21: Collections) to include a document by reference.

Both the <affectedDocument> element and the <relatedDocument> element should only be used in the preface to an amendment.

The referencing elements described above support a basic set of attributes:

	<a>	<ref>	<rref>	<mref>	<affectedDocument>	<relatedDocument>
@href	Opt.	Req.	-	-	Req.	Req.
• URI pointer to a resource.						
@target	Opt.	-	-	-	-	-
• As with HTML, the browser window in which to display the referred to item.						
@from	-	-	Req.	-	-	-
• An id reference to the initial provision in a range of referenced provisions.						
@upTo	-	-	Req.	-	-	-
• An id reference to the last provision in a range of referenced provisions.						

In addition to these attributes, there are two additional attributes which act as references when used on other elements:

@alternativeTo — (Optional) This attribute is an id reference to another element that the element having this attribute is an alternative copy to.

`@refersTo` — This attribute is an id reference to an element in the metadata that is part of an ontology. This attribute is required on elements that make reference to the ontology. This topic will be discussed in *Chapter 20: Ontologies*.

Reference Metadata

In addition to the reference tags, Akoma Ntoso provides a mechanism in the metadata to record other types of internal references — using the `<references>` container in the `<meta>` element. Much of this container is dedicated to ontologies and will be discussed in *Chapter 20: Ontologies.*

In addition to the ontology references (which all start with the prefix *TLC*), there are a few other references which optionally can be added to the <references> container:

`<original>` — This element provides a reference to the original version of this document.

`<activeRef>` — This element provides a reference to a document that this document modifies (amends).

`<passiveRef>` — This element provides a reference to a document that makes modifications to this document.

`<attachmentOf>` — This element provides a reference to a document that this document is attached to.

`<hasAttachment>` — This element provides a reference to a document that is attached to this document.

`<jurisprudence>` — This element provides a reference to a document that provides jurisprudence (a legal basis) to this document.

There is no pre-defined order for reference elements in the `<references>` container. All elements may occur zero or more times, although `<original>` will most likely occur at most once.

Passive and active references will be discussed with amending in the upcoming chapters.

Maintaining References

References between documents and provisions are very useful in that they facilitate navigation. However, for this to work well, it is necessary that references be accurate and be maintained throughout the lifecycle of a document. When a provision is moved or renumbered, the references to it should update accordingly. Also, when a provision is amended, the references should be examined to

ensure that the reference remains pertinent to the modified language.

An editor should be able to provide assistance when managing references, although providing a fool-proof solution can be challenging.

One way to provide reliable references is to build some degree of redundancy into the referencing mechanism. This can be done by tying a `<ref>` to a `<TLCReference>` in the metadata and having both point to the provision or document in question. The `<ref>` element uses its `@href` attribute to point to the current location (the `@eId`) of the provision while the `<TLCReference>` uses its `@href` to point to the provision itself (version independent `@wId` or `@GUID`). The `@refersTo` attribute on the `<ref>` ties it to the `<TLCReference>`. Should the version dependent reference on the `<ref>` become broken, a pointer to the current version can be established by finding the object via the version independent reference on the `<TLCReference>`.

The code sample below shows how to use this technique to create a redundant pointer that can recover from moves and renumbers of the provision:

```
<references>

  <TLCReference
    eId="ref1"
    href="#1234-5678"
    showAs="Section 5"/>

</references>

…

… <ref
    href="#sec_5"
    refersTo="#ref1">section 5</ref> …

…

<section eId="sec_5" GUID="1234-5678">

   …

</section>
```

Alternatively, you could use the passive modification approach described in Chapter 17: Passive Modifications to manage renumbering.

In the next chapter, we will start to look at amending, one of the most important parts of the entire legislative cycle.

Akoma Ntoso Pocket Guide

Chapter 16: Amending Basics

Amending is one of the most important, if not the most important, aspect of the entire legislative cycle. Documents are created and then modified using a very precise and tightly controlled update process. How all the changes are managed is the topic of the next four chapters.

We will explore a number of different aspects of Akoma Ntoso's change management system:

- Temporal modelling of the changes
- The difference between active and passive modifications
- Active modifications and amendments
- Passive modifications and redlining
- Quoted text and structures

Temporal Modelling

Akoma Ntoso provides extensive support for modelling the temporal aspects of provisions found within the legal text. This is an advanced topic that requires substantial analysis of the legal text to build a metadata model that can describe which provisions

are in effect, in force, or have been repealed and at what time and under which conditions.

The <temporalData> container is found in the <meta> element and is used to describe time periods that are fundamental to the modelling of the temporal aspects of the provisions in a document.

```
<temporalData source="{source}">
  <temporalGroup eId="{identifier}">
     <timeInterval refersTo="{ontologyRef}"
        [start="{eventRefRef}"]
        [end="{eventRefRef}"]
        [duration="{duration}"]/>
     …
  <temporalGroup>
  …
</temporalData>
```

Optional attributes are shown in the example above using square brackets.

This structure is used to define time periods (as temporal groups) that are then referred to either in the analysis (described next) or by elements in the document body. A time period will usually be defined as a single time interval, but can also be defined as multiple non-contiguous time intervals.

Event references are defined in the lifecycle portion of the metadata. The `@start` and `@end` attributes are references to events defined this way.

A time interval should either be specified with a `@start` and a `@duration` or with a `@start` and an `@end`. If the time period is open ended (includes now), then only specify the `@start`.

The `@period` attribute, available on numerous elements including the various modification types (described next), metadata notes, and most other non-metadata elements is used to refer to a time period described as a `<temporalGroup>` that the element applies to.

Active vs. Passive Modifications

There are two primary types of changes, known as modifications, that Akoma Ntoso supports:

1. When one document changes another document, those changes are called active modifications. There are two common instances of active changes in legislative documents. The first are acts that affect statutes or codes. The second are amendments that affects bills.

2. When a document contains changes within itself, those types of changes are called passive modifications. Often in jurisdiction having Common Law traditions, passive modifications are shown as *redlining* (or change tracking).

Analysis Container

The `<analysis>` container in the `<meta>` section is a description or characterization of the content of the text of the document. This container attempts to model the language of the document, as it relates to amending, in a machine processable way. It is usually created or updated after a document has been edited, although Akoma Ntoso does not dictate when or how this metadata is produced.

The `<analysis>` container contains information about *active* and *passive* amendments (among other things). There are two sub-containers within the `<analysis>` container that contain information about modifications:

`<activeModifications>` — This element contains a list of modifications that the document makes to other documents.

<passiveModifications> — This element contains a list of modifications implemented in the document.

Both of these containers have the same structure and can contain the same types of elements, each describing a different modification either made by or contained within this document.

The following types of modifications are supported:

<efficacyMod> – This element changes a time period the text is in effect.

<forceMod> – This element changes a time period the language is in force.

<legalSystemMod> – This element changes the legal system behind the language.

<meaningMod> – This element changes the meaning of the language.

<scopeMod> – This element changes the scope of applicability of the language.

<textualMod> – This element is a change to the text. This is the most common type of modification.

Each of these six modifications must be further specified with the @type attribute which

characterizes the details of the modifications. The following table summarizes the types for each modification:

<efficacyMod>	<forceMod>
entryIntoEfficacy endOfEfficacy inapplication retroactivity extraEfficacy postponementOfEfficacy prorogationOfEfficacy	entryIntoForce endOfEnactment postponementOfEntryIntoForce prorogationOfForce reEnactment unconstitutionality
<legalSystemMod>	**<meaningMod>**
staticReference implementation ratification application legislativeDelegation deregulation conversion expiration reiteration remaking republication coordination	variation termModification authenticInterpretation
<scopeMod>	**<textualMod>**
exceptionOfScope extensionOfScope	repeal substitution insertion replacement renumbering split

In addition to the `@type` attribute, each modification can contain a number of other attributes:

`@eId`, `@wId`, `@GUID` – These attributes are the normal identifiers which are used to refer to a modification. Typically, only the `@eId` will be used.

`@refersTo` – This attribute is a reference to the ontology.

`@exclusion` – This attribute is a Boolean that is used to indicate that the action of the modification should be applied to all the partitions of the document except the destination explicitly mentioned.

`@incomplete` – This attribute is a Boolean that can indicate that the analysis that produced the modification may not be complete.

`@period` – This attribute is a reference to a `<temporalGroup>` and is used to define a time period that this modification applies to.

`@status` – This attribute is a set of values, ordinarily used to describe a discrepancy between the manifestation and the expression, may also be used here to describe the status of the modification if the analysis may not be correct or current.

In addition, there are several elements that each modification uses to describe the modification:

`<source>` – Provides a reference to the provision that expresses the modification.

`<destination>` – Provides a reference to the provision(s) that is affected by the modification.

`<old>` – (Optional) Provides a reference to an element containing the text quoted[2] as it was before it was changed. This element is only defined for textual modifications.

`<new>` – (Optional) Provides a reference to an element containing the text quoted as it is to become. This element is only defined for textual modifications.

`<previous>` – (Optional) Provides a reference to a provision in the prior expression of the document that is affected by a renumbering. This element is only defined for textual modifications.

This is a partial list. There is a further set of elements that are part of the temporal and efficacy modelling and these are described in the next section.

Efficacy and Force Modelling

In addition to the basic modelling of modifications, further analysis can be applied to describe additional aspects of when the modification is in effect and/or in force.

[2] Quoted text and structures will be discussed in an upcoming section

These elements can appear at most once. Use the `@period` attribute with the elements that specify time periods to refer to a temporal group in the `<temporalGroup>` container.

`<application>` – Specifies a period that the modification applied.

`<duration>` – Specifies a duration that a modification applies.

`<efficacy>` – Specifies a time period that the modification is in effect.

`<force>` – Specifies a time period that the modification is in force (may be a shorter interval within the period that the modification is in effect).

`<condition>` – Specifies an open set of conditions that may affect the modification. Akoma Ntoso does not attempt to either further specify or interpret the conditions.

`<domain>` – Restricts the domain of a meaning or scope modification. Akoma Ntoso does not attempt to further specify or interpret the domain.

Note: It is possible to specify the `@period` on the modification itself or on one of the child elements that define a time period. Use the `@period` attribute on the modification element only when further analysis to yield more granular metadata is not possible or not available.

With the metadata for amending covered, it's now time to take a look at how amendment text itself is modelled in Akoma Ntoso. The next chapter will start the topic by looking at passive amendments.

Akoma Ntoso Pocket Guide

Chapter 17: Passive Modifications

Passive modifications are changes within the current document. The previous section described how they, along with active modifications, are modelled within the <analysis> container of the metadata. This section will cover how they are modelled within the text of the document.

Insertions and Deletions

Similar to HTML, Akoma Ntoso offers two tags to represent insertions and deletions within the text of a document. Both are simple inline elements:

`<ins>` – Shows text that is being inserted.

`<del>` – Shows text that is being deleted (or left out).

These elements should be assigned `@eId` values as they will be referred to by these values by the passive modifications in the `<analysis>` container.

Omissions

Text that has been removed by a prior amendment can be recorded as having once existed using a tombstone element:

`<omissis>` – Specifies that text at this location has been removed, either by amendment or merely for presentation. The content of this element typically will denote the omission using language such as '…' or *omissis*.

Alternate Representation of Changes

The `<ins>` and `<del>` elements provide a common and easily understand method for showing insertions and deletions within text. However, they have limitations in that they are inline elements and cannot represent all possible insertions and deletions, particularly of more complex structures.

To address this limitation, many XML editors rely on a different mechanism to represent insertions and deletions. Pioneered by SoftQuad's (now JustSystems) XMetaL, many XML editors use a set of XML processing instructions. A single deletion processing instruction denotes text that is deleted and a wrapping pair of processing instructions is placed around text that is inserted. This approach is, more or

less, the common approach to solving this problem for editing. It can, and has been, successfully used with Akoma Ntoso.

As most (or all) XML editors that support track changes don't provide much option for configuring how change tracking is handled, the editor's intrinsic mechanism that relies on processing instructions should be used.

However, as Akoma Ntoso specifies the inline <ins> and <del> tags for showing changes, any exported manifestation intended for external consumption should convert the processing instructions into the <ins> and <del> inline tags, whenever possible.

For instance, the following text used within the drafting environment:

```
<p> If a <?delete text="majority"?>
<?insertStart?>plurality<?insertEnd?> of
the parties to the arbitration agreement
are citizens of a State or States that have
ratified or ….</p>
```

becomes:

```
<p> If a <del>majority</del>
<ins>plurality</ins> of the parties to the
arbitration agreement are citizens of a
State or States that have ratified or
….</p>
```

when exported for external consumption.

Special care should be taken to ensure that the resulting transformation yields a document that is valid with the Akoma Ntoso schema as the processing instruction markers may not correspond one-for-one with the inline Akoma Ntoso `<ins>` and `<del>` elements.

With passive modifications covered, it's time to focus on active modification. However, before we can get too far into this subject, we need to understand how quoting text and structures work. The next chapter will cover the various elements that exist to handle quotes.

Akoma Ntoso Pocket Guide

Chapter 18: Embedded and Quoted Structures

There are many reasons that text may be quoted within legislation and enacted laws. Often it is to merely express a textual passage that isn't part of the primary text. More often, it is to refer to or describe text that is being amended.

Akoma Ntoso provides two sets of elements to handle these cases:

`<embeddedText>`, `<embeddedStructure>` – These elements contain either simple text or a more complex structure that is being stated within the text of the document.

`<quotedText>`, `<quotedStructure>` – These elements contain either simple text or a more complex structure that is part of an active modification that is being expressed.

While these four elements are used similarly, their use varies based on where the element is to be used and what it contains – quoted elements for amending language and embedded elements everywhere else.

There are four optional attributes that can be used with these elements:

`@startQuote` – Specifies the character to use to start the quote. There are many characters which may be used, depending on local tradition. In English-speaking countries, this is often the single start quote (‘) or the double start quote (“).

`@endQuote` – Specifies the character to use to end the quote. There are many characters which may be used, depending on local tradition. In English-speaking countries, this is often the single end quote (’) or the double end quote (”).

`@inlineQuote` – Specifies the character to use at the beginning of each paragraph within the quote. Some jurisdictions require that the start quote be repeated for each paragraph and this is a common grammatical convention. For example, when quoting a section of the U.S. Code:

“§701b–8. Submission of report by Chief of Engineers

“It is declared to be the policy of the Congress that the following provisions shall be observed:

“No project or any … conformity with existing law.”

`@href` – (`<embbededText>` and `<embeddedStructure>` only) Specifies a reference to the origin of this quote.

`@for` – (`<quotedText>` and `<quotedStructure>` only) Specifies an id reference to the element that this is a modification of.

Note: There is a general principle in legislative markup that generated text is a bad practice to be avoided at all costs. This is primarily due to the difficulty of handling amending with generated content. For this reason, you may choose to, for internal processing, to handle the quote marks as inline characters within the text content. In some respects, the styling of the elements with CSS will also be easier. However, this approach does require additional processing of the quoted elements when applying amending language. Also, when exporting a document for external consumption, you might consider producing a manifestation that transforms the result to use the quote mark attributes instead.

This chapter laid some of the groundwork for active modifications. The next chapter will cover active modifications and some of the amendment styles that they support.

Akoma Ntoso Pocket Guide

Chapter 19: Active Modifications

Active modifications are perhaps the heart of legislative drafting – describing additions and other changes to existing law. An earlier chapter described how this information is captured in the <analysis> container. This chapter will now focus on how this information is modelled within the language of the bill (or amendment) itself.

Modifications

Active modifications are expressed within the `<mod>` inline element. This element will contain the text describing the modification (sometimes called the *action line*) and might contain one or more quoted content elements (`<quotedText>` or `<quotedStructure>`). The quoted content contains either text to be changed or the resulting text.

Amendments can be grouped together, if necessary, using the `<mmod>` element.

Amendment Styles

There are several different amendment styles and all can be handled by Akoma Ntoso:

1. Cut and Bite – this is the most common type of amending language for bills. It specifies specific changes to be made to the text in the form of insertions and deletions. The location in the text to be changed is either a physical reference to a page and line number (usually in amendments) or a logical reference to a hierarchical level or section (usually in bills).

```
<mod>
  On <ref>page 1, line 10</ref>,
  strike
  <quotedText>majority</quotedText>
  and insert
  <quotedText>plurality</quotedText>
</mod>
```

2. Restatement in Full – this is the amendment style often found in state legislation in the United States when amending codes or statutes. (Cut and bite is still used for bill amendments). With this style, the section (most often) that is to be modified is restated entirely with the changes shown (sometimes) as strikes and insertions.

```
<mod>
  Section 1234 is amended to read:
  <quotedStructure>
    <section>
      <num>1234</num>
      <content>
        <p>If a <del>majority</del>
        <ins>plurality</ins> of the
        parties to the arbitration
        agreement are citizens of a
        State or States that have
        ratified or ….</p>
      </content>
    </section>
  </quotedStructure>
</mod>
```

While restatement in full is obviously more verbose, it is more transparent in that it more clearly shows that context of the amendment and it is easier to automate the engrossing of amendments.

3. Side-by-side or double-column – this is the amendment style often found in Europe. With this style, the original next of a provision is shown in the left column and the proposed text is shown on the right with changes sometimes denoted in some way.

```
<mod eId="mod_1">
 <quotedStructure eId
="mod_1__qst_1"
    refersTo="#initialVersion">
  (2) In numerous resolutions, the
  European Parliament has expressed
  its concern at the destruction of
  forests and the consequences for
  forest peoples.
 </quotedStructure>
 <quotedStructure id=" mod_1__qst_2"
    refersTo="#newVersion">
  (2) In numerous resolutions, the
  European Parliament has expressed
  its concern at the destruction of
  forests and the consequences for
  forest peoples<ins>, in particular
  indigenous peoples</ins>.
 </quotedStructure>
</mod>
```

In the next chapter, we turn our attention away from amending and focus on how to tag various elements in the document with references to an ontology that you define.

Akoma Ntoso Pocket Guide

Chapter 20: Ontologies

Increasingly there is a desire to tag the information within legislation to make it easier for transparency organisations and search engines to determine what the legislation concerns. This tagging requirement brings us to the subject of ontologies.

An ontology is an information model that describes the entities in a particular domain or subject area. In legislation, ontologies might be designed to describe the various entities and stages in the legislative process as well as the numerous topics that legislation might cover.

Top Level Classes

Akoma Ntoso provides a way to reference such ontologies from within the document. The `<references>` container can be populated with references to the various *Top Level Classes* which comprise the ontology.

There is a set of elements to characterize items in the ontology as people, places, things, processes, etc. These elements all begin with the prefix `TLC:`

<TLCPerson> —

<TLCOrganization> —

<TLCConcept> —

<TLCObject> —

<TLCEvent> —

<TLCLocation> —

<TLCProcess> —

<TLCRole> —

<TLCTerm> —

<TLCReference> —

Each of these elements use the following attributes:

@eId – This attribute is used to identify the element. Elements in the document that refer to a TLC element will use the @refersTo attribute to point to this element using the syntax #{@eId} where {@eId} is the @eId attribute value.

@href – This attribute is a URI that uniquely identifies the entity in your ontology. Similar to Namespace URIs, the URI need not resolve, but it is a

good idea that it point to an information page about the entity.

`@showAs` – This attribute provides a descriptive name to the entity in the ontology.

`@shortForm` – (Optional) This attribute provides a shorter alternative name for the entity in the ontology.

Referring to the Ontology

The ontology is used in a document to tag items by wrapping elements around text that refers to entities in the ontology and then associating that element with the entity in the ontology. There are a few attributes that are used to refer to entities in the ontology:

`@refersTo` – This attribute is the general mechanism that is used to make a reference to an entity in the ontology. It is an internal id reference of the form `#{eId}` where the eId value is the `@eId` attribute value of the element corresponding to the ontology entity in the `<references>` container.

`@source` – This attribute is also used to refer to entities in the ontology, but for one specific reason – to identify the source of information for that container. The source might be a person, an

organisation, a process, or even an application. To use the @source attribute, as required, there will need to be a corresponding entity in the ontology.

Managing the Ontology.

It is quite possible, even likely, that an ontology will be far larger than the needs of any specific document. As Akoma Ntoso requires that entries in an ontology that are referred to by the document be held in the `<references>` container of the document's metadata, a strategy should be devised to ensure that a document has the necessary entries in the `<references>` section and that they are added, updated, and removed as necessary.

For some small ontologies, it might be acceptable to define the entire ontology within the `<references>` section, but in general, this will be overkill and will become too cumbersome.

Tagging Elements

In order to tag elements and relate them to the ontology, Akoma Ntoso provides an extensive set of inline elements that can be used:

`<person>` — This element is to be used to identify a person and corresponds to a `<TLCPerson>` in the ontology.

<organization> — This element is to be used to identify an organisation and corresponds to a <TLCOrganization> in the ontology.

<concept> — This element is to be used to identify a concept and corresponds to <TLCConcept> in the ontology.

<object> — This element is to be used to identify an object and corresponds to <TLCObject> in the ontology.

<event> — This element is to be used to identify an event and corresponds to <TLCEvent> in the ontology.

<location> – This element is to be used to identify a location and corresponds to <TLCLocation> in the ontology.

<process> — This element is to be used to identify a process and corresponds to <TLCProcess> in the ontology.

<ref> — This element, already covered in the chapter on references, is to be used to identify a reference and corresponds to <TLCReference> in the ontology.

<role> — This element is to be used to identify a role and corresponds to <TLCRole> in the ontology.

<term> — This element is to be used to identify a term and corresponds to <TLCTerm> in the ontology.

<quantity> — This element is to be used to identify a quantity and corresponds to <TLCQuantity> in the ontology.

<entity> — This element plays a generic role and can be used to define your own type of tag using the @name attribute.

All these elements require the use of the @refersTo attribute to associate the element to its top level class element in the ontology. In addition to these elements, there is an additional set of elements that can be used to tag other items. For these elements, the use of the @refersTo attribute is optional.

<date> — This element is used to tag a date. The @date attribute is required and expresses the date in a normalized form using the XML Schema syntax of YYYY-MM-DD or, when more precision is required YYYY-MM-DDThh:mm:ss(zzzz).

`<time>` — This element is used to tag a time. The `@time` attribute is required and expresses the date in a normalized form using the format hh:mm:ss.

`<def>` — This element is used to tag a definition.

Akoma Ntoso Pocket Guide

Chapter 21: Collections

Sometimes it is necessary to create a document that is a composite document, made up of different parts. For these cases, collections are used.

There are three specific collection documents:

`<amendmentList>` — a collection of changes.

`<documentCollection>` — a more generic composition of other documents.

`<officialGazette>` — a specific composition of documents that forms the official publication of the legislative body.

Like all document types, all three of these document types require the use of the `@name` attribute to further define the document type.

All collection documents use the `<collectionBody>` element as their body element. A `<collectionBody>` is composed of one or more <component> elements.

Each `<component>` is a constituent part of the document and can either be included inline or by

reference. Any of the twelve document types, including other collection documents can be included within a <component> element.

In addition to all the document types, components can include other documents by reference using the `<documentRef>` element with the `@href` attribute.

Components can also contain `<num>`, `<heading>`, `<subheading>`, `<toc>`, and `<interstitial>` elements. The `<interstitial>` element is similar to the `<content>` element and can be used to place arbitrary text between components.

Amendment Lists

A special case of the document collection is the amendment list. Depending on your jurisdiction, an amendment may either be regarded as a single change or as a collections of changes (or instructions). In Akoma Ntoso, a single change is contained in an `<amendment>` document while a set of changes are represented by the `<amendmentList>` document type.

Individual changes, contained in `<amendment>` documents, can be included inline within an `<amendmentList>` without requiring a separate file for each individual amendment. While this

representation does work, it is a bit bulky for the case where individual amendments (or instructions) are never treated alone.

For jurisdictions that compose amendment lists out of individual amendments, choosing whether to include the amendment inline or by reference comes down to whichever approach is most expedient.

Akoma Ntoso Pocket Guide

Chapter 22: Portions

Some legal documents, especially codes, can become very large – too large to be manageable for tasks like editing. Akoma Ntoso addresses this issues with a special type of document called a `<portion>`.

A portion is an extraction of a master document and contains its own metadata and a `<portionBody>` element as its body.

The `<portion>` document type, unlike other document types, does not use the `@name` attribute. Instead, it uses the `@includedIn` attribute which is an id reference to the part of the master document that the portion is an extraction of.

The `<portionBody>` simply contains an extraction from the master document. The following elements can be included in a portion:

```
<administrationOfOath>, <rollCall>,
<prayers>, <oralStatements>,
<writtenStatements>,
<personalStatements>,
<ministerialStatements>,
<resolutions>, <nationalInterest>,
<declarationOfVote>, <comunication>,
```

<petitions>, <papers>,
<noticesOfMotion>, <questions>,
<address>, <proceduralMotions>,
<pointOfOrder>, <adjournment>,
<debateSection>, , <container>,
<clause>, <section>, <part>,
<paragraph>, <chapter>, <title>,
<article>, <book>, <tome>, <division>,
<list>, <point>, <indent>, <alinea>,
<rule>, <subrule>, <proviso>,
<subsection<subpart>, <subparagraph>,
<subchapter>, <subtitle>,
<subdivision>, <subclause>, <sublist>,
<level>, <transitional>, <hcontainer>,
<speechGroup>, <speech>, <question>,
<answer>, <other>, <scene>,
<narrative>, <summary>, <recital>,
<recital>, <citations>, <citation>,
<longTitle>, <formula<coverPage>,
<preface>, <preamble>

Note: A portion body cannot contain a few useful elements such as <table> or <toc>. To work around this limitation, you can wrap the unsupported element in a supported element such as .

When extracting a portion, the relevant metadata should be extracted from the master document to ensure that the portion document is a whole document and that referential integrity is maintained.

Also, internal id references to items outside the bounds of the portion should be converted to be external id references.

In all likelihood, a `<portion>` will never be stored in a repository. Instead, it's a transitory document that is created by extraction for the use of a tool and then used to update the master document when being saved rather than being stored itself. As such, it's a facility that exists for the convenience of tools more than a mechanism for modelling a document.

When stitching a portion back into the master document, a careful synchronization should be performed to once again ensure the referential integrity of the document and to ensure that the metadata is properly formed.

Akoma Ntoso Pocket Guide

Chapter 23: Next Steps

Akoma Ntoso provides a comprehensive XML-based legislative markup language that is flexible enough to map to virtually all legal traditions around the world. It has been extensively tested against numerous scenarios and is in the process of being implemented in a number of high-profile jurisdictions.

This primer has provided a comprehensive overview of the Akoma Ntoso element set, but there remains more to learn. The element set for debates, debate reports, gazettes, and several other document types can be learned as needed. They all follow the same principles as the elements covered in this primer.

If you are curious about how suitable Akoma Ntoso is for your environment, the next step is to try and model all your documents using the Akoma Ntoso element set. This can start off as a set of manually converted documents, but at some point you might want to try to produce an automated converter in order to get comprehensive coverage of all the documents used in your jurisdiction – including historical documents that might contain forgotten

practices which, nonetheless, will have to be represented.

When it comes to developing tools that support Akoma Ntoso, the good thing about having a standard is that it creates a basis for an entire ecosystem of applications that will work together relatively seamlessly. My company, Xcential Legislative Technologies, has developed an editor and other tools, and other vendors are doing likewise. These tools promise far greater sophistication and affordability than the custom tools of the past.

If you wish to dive into the deep end and start developing your own tools, a worldwide community of Akoma Ntoso practitioners already exists and can be a valuable resource for lessons learned and other experiences.

The primary resources at your disposal can be found at:

- The Official OASIS website: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=legaldocml
- The GitHub repository: https://github.com/oasis-open/legaldocml-akomantoso
- The Akoma Ntoso website: http://akomantoso.org

Also, check http://xcential.com for future tool and service announcements from Xcential Legislative Technologies in the area of Akoma Ntoso.