Story Workbench
Format Specification
Version 1.2.0

Mark A. Finlayson
(markaf@alum.mit.edu)

October 21, 2012

Contents

1 Purpose of this Document

This reference document outlines the syntax of a Story Workbench annotation file. It is provided to allow clients to write their own XML handlers to parse Story Workbench data.

The document is split into three sections. Section 2 defines a DTD (XML Document Type Definition) that applies to all version 1.0 Story Workbench data files. The DTD defines the overall XML structure of the files. Section 3 defines the most common annotation layers supported by the workbench. Each layer governs the format of the `PCDATA` inside the `desc` tags for that layer. Section 4 defines the most common metadata types supported by the workbench. Each metadata type governs the format of the `PCDATA` inside the `metaDesc` tags for that metadata type.

2 General DTD

Listing 1 defines the Document Type Definition for version 1.0 story workbench files.

```
1  <!ELEMENT story    (rep+,metaRep*)          -- root tag of a .sty file  -->
2  <!ELEMENT rep      (param*,factory?,desc*) -- annotation layer          -->
3  <!ELEMENT desc     (#PCDATA)                -- individual annotation    -->
4  <!ELEMENT factory  (param*)                 -- automatic annotator      -->
5  <!ELEMENT param    (param*)                 -- configuration parameter  -->
6  <!ELEMENT metaRep  (metaDesc*)              -- set of metadata          -->
7  <!ELEMENT metaDesc (#PCDATA)                -- piece of metadata        -->
8
9  <!ATTLIST story    ver    CDATA #IMPLIED "1.0" -- format version       -->
10 <!ATTLIST rep      id     CDATA #REQUIRED      -- layer id             --
11                    ver    CDATA #IMPLIED "1.0" -- layer version        -->
12 <!ATTLIST desc     id     CDATA #REQUIRED      -- annotation id number --
13                    len    CDATA #REQUIRED      -- annotation length    --
14                    off    CDATA #REQUIRED      -- annotation offset    -->
15 <!ATTLIST factory  id     CDATA #REQUIRED      -- id of the factory    -->
16 <!ATTLIST metaRep  id     CDATA #REQUIRED      -- metadata id          -->
17 <!ATTLIST metaDesc descID CDATA #REQUIRED      -- target annotation id --
18                    repID  CDATA #REQUIRED      -- target layer id      -->
```

Listing 1: Story File DTD

This DTD covers the general structure of the XML document, but does not cover these additional restrictions on the elemenets:

1. The `desc` children of certain `rep` elements may reference, in their `PCDATA` sections, the numeric `id` of `desc` children in other parts of the document. If this occurs, it means that those layers of representation depend on other layers. If this is the case the `rep` element corresponding to the dependent layer must occur after the `rep` layers on which it depends.

2. Each `param` element may have only one attribute; the name of the attribute, and its allowed values, is specific to the parent element of the `param`.

2.1 Attribute Syntax

i. `ver` — All version attributes should be a string matching the following BNF, which generally looks like `1.0`, `2.0.2`, or `3.1.4.somequalifierstring`.

```
Symbol           Expression
version    ::=  major('.'minor('.'micro('.'qualifier)?)?)?
major      ::=  digit+
minor      ::=  digit+
micro      ::=  digit+
qualifier  ::=  (alpha|digit|'_'|'-')+
digit      ::=  [0..9]
alpha      ::=  [a..zA..Z]
```

ii. `id` — The identifier attribute of the `rep`, `factory`, and `metaRep` elements are unique strings that following a reverse-namespace naming convention, such as `com.xyz.myworkbenchcode`. They match the following BNF:

```
Symbol       Expression
id      ::=  part(.part)*
part    ::=  (alpha|digit|'_'|'-')+
digit   ::=  [0..9]
alpha   ::=  [a..zA..Z]
```

iii. `id` — The identifier attribute for the `desc` element is a non-negative integer that is unique across all `desc` elements in the document.

iv. `descID` — This attribute on a `metaDesc` element references a `desc id` attribute. The referenced `desc` element may or may not actually exist in the document (i.e., it may have been deleted).

v. `repID` — This attribute on a `metaDesc` element references a `rep id` attribute. The referenced `rep` element may or may not actually exist in the document (i.e., it may have been deleted).

vi. `len, off` — Length and offset attribute should non-negative integers.

3 Representation Formats

Each representation section following describes the purpose of that annotation layer, gives a syntax for the encoding of the `PCDATA` inside the `desc` elements for that layer, and gives example XML of that layer. Because later layers build on previous layers, and reproducing all the building blocks for each successive layer would be prohibitively long (and difficult to understand), many of the examples build upon previous examples by using the same text and referencing `id` numbers in other figures. When portions of XML are elided for this purpose, an ellipsis (. . .) is shown. Also, for ease of exposition, no `factory` or `param` elements are included, except in exceptional cases where they are important to understand how the XML should be parsed.

3.1 Characters

The Character representation is the root layer of annotation in each story file. It contains the actual text of the story, and the `off` and `len` numbers in each `desc` element in the document refer to a span of characters with specified offset and length in the character layer. An example minimal story file is shown in Listing 2.

```
1  <?xml version="1.0" encoding="UTF-8"?>@@
2  <story>
3    <rep id="edu.mit.story.char">
4      <desc id="0" len="41" off="0">
5  This is a
6  story about a man
7  named Jed.
8
9  </desc>
10   </rep>
11 </story>
```

Listing 2: A minimal story file

The first line is the standard XML declaration. The XML document contains one root tag, `story`, which has a single child `rep` element. Each `rep` element requires an id, which identifies the layer it encodes. Here the `id` is `edu.mit.story.char`. In version 1.0 of the story workbench format , the character layer may have only a single `desc` child element, which must have an `id` of 0. The offset of this root annotation element must be 0, and the length is equal to the number of characters in the file.

The character block is the actual text of the story, encoded verbatim, except for standard XML escape sequences representing prohibited characters. Line breaks are encoded in UNIX style, with a single character, the linefeed (`lf` or `\n`). For convenience's sake, inside the XML document, the text has a single linefeed character appended to both its start and end; this has the effect of having the text start on a newline in the XML file (line 5 in the file) and have the closing tag of the `desc` element on a new line (after line 8). Therefore, to extract the exact text from the XML file, one need only copy the full line after the opening `desc` tag with `id` of 0, to the full line just before the closing desc tag. So the actual full text of the file shown in Listing 2 is shown in Figure 1.

```
|This␣is␣a␣¶
|story␣about␣a␣man␣¶
|named␣Jed.¶
|
```

Figure 1: Actual text of the minimal story file. Every space is shown with the ␣ symbol, every newline character is shown with ¶, and every line is prefaced by a | symbol. There are four lines (the last one is empty) and forty-one characters, which includes spaces and newline characters.

3.2 Text

All layers of annotation in the story file depend on the character representation, and so the character representation comes first. The next most basic layer of annotation tells us where comments versus actual meaningful text appear in the text of the story. For example, the Story Workbench story editor admits Java-style comments, as shown in Figure 2.

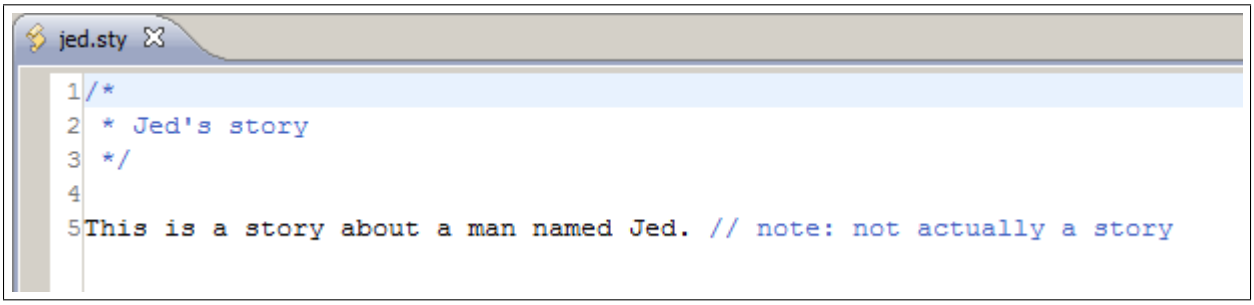

Figure 2: How comments appear in the Story Editor

The BNF describing the `PCDATA` of the Text `desc` elements is as follows:

Symbol		Expression
PCDATA	::=	"TEXT" | "SINGLE" | "MULTI"

Table 1: BNF for Text Representation

The meaning of each constant is straightforward. `TEXT` means actual text to be analyzed with annotations, whereas `SINGLE` and `MULTI` mean single- and multi-line comments, respectively. An example of this encoding is shown in Listing 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
    <desc id="0" len="91" off="0">
/*
 * Jed's story
 */

This is a story about a man named Jed. // note: not actually a story
</desc>
  </rep>
  <rep id="edu.mit.story.text">
    <desc id="68" len="21" off="0">MULTI</desc>
    <desc id="69" len="41" off="21">TEXT</desc>
    <desc id="70" len="29" off="62">SINGLE</desc>
  </rep>
</story>
```

Listing 3: Example Text Encoding

3.3 Tokens

The Token layer is one of the most fundamental layers of annotation in the whole document. Most, if not all, layers of annotation depend on the Token layer in one way or another. The Token encoding is quite straightforward. Tokens are unbroken spans of characters in the original document that do not contain whitespace. The `PCDATA` inside of a Token `desc` element is merely the tokenized form of the token in the text. The Story Workbench uses the Penn Treebank tokenization conventions. This means that, for most tokens, the tokenized form is identical to the surface form of the token (i.e., how it actually appears in the text). But some tokens, such as brackets, are tokenized slightly differently: for example, parentheses `(` and `)` are encoded as `-LRB-` and `-RRB-`, respectively.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <story>
3    <rep id="edu.mit.story.char">
4  This is a story about a man named Jed.
5  </desc>
6    </rep>
7  ...
8    <rep id="edu.mit.parsing.token">
9      <desc id="16" len="4" off="0">This</desc>
10     <desc id="17" len="2" off="5">is</desc>
11     <desc id="18" len="1" off="8">a</desc>
12     <desc id="19" len="5" off="10">story</desc>
13     <desc id="20" len="5" off="16">about</desc>
14     <desc id="21" len="1" off="22">a</desc>
15     <desc id="22" len="3" off="24">man</desc>
16     <desc id="23" len="5" off="28">named</desc>
17     <desc id="24" len="3" off="34">Jed</desc>
18     <desc id="25" len="1" off="37">.</desc>
19   </rep>
20 </story>
```

Listing 4: Example Token Encoding.

3.4 Collocations / MWEs

The Multi-word Expression (MWE) annotation layer (also known as the Collocation layer) allows individual tokens to be bound into groups for later syntactic or semantic processing. Good examples of MWEs are phrasal verbs ("look up", "let go"), proper nouns ("Jed Clampett"), or idioms ("kick the bucket").

Each MWE `desc` element's `PCDATA` is expressed as a comma-delimited list of token `desc` `id` numbers. An example of this is shown in Listing 5, where there are three collocations: "Jed Clampett" (tokens 2 and 3); "let... go" (tokens 4 and 6); and "Beverly Hills" (tokens 8 and 9). Note, as in the case of "let... go", MWEs do not need to be continuous.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <story>
3    <rep id="edu.mit.story.char">
4      <desc id="0" len="41" off="0">
5  Jed Clampett let him go to Beverly Hills.
6  </desc>
7    </rep>
8  ...
9    <rep id="edu.mit.parsing.token">
10     <desc id="2" len="3" off="0">Jed</desc>
11     <desc id="3" len="8" off="4">Clampett</desc>
12     <desc id="4" len="3" off="13">let</desc>
13     <desc id="5" len="3" off="17">him</desc>
14     <desc id="6" len="2" off="21">go</desc>
15     <desc id="7" len="2" off="24">to</desc>
16     <desc id="8" len="7" off="27">Beverly</desc>
17     <desc id="9" len="5" off="35">Hills</desc>
18     <desc id="10" len="1" off="40">.</desc>
19   </rep>
20   <rep id="edu.mit.parsing.colloc">
21     <desc id="11" len="12" off="0">2,3</desc>
22     <desc id="13" len="10" off="13">4,6</desc>
23     <desc id="12" len="13" off="27">8,9</desc>
24   </rep>
```

Listing 5: MWE Representation

3.5 Representation Building Block: Segments

Before moving on to higher-level layers of annotation, we will define here a common encoding syntax, which will be called "segments". A "segment of annotations" for a particular layer of annotations means an unbroken run of annotations in that layer. Take, for example, a segment of tokens in the text displayed in Listing 5, starting from the token "Clampett" to the token "go". The segment will contain five tokens: "Clampett let him go", and would be encoded as shown in line 1, namely, a tilde-delimited list of `desc id` numbers. The segment defined by the two tokens "Beverly Hills" would be encoded as on line 2.

A segment set is a set of non-overlapping segments, and is a comma-delimited list of segment encodings. So the segment set comprised of the two above-mentioned segments would be encoded as on line 3.

```
1 3∼4∼5∼6          // segment "Clampett let him go"
2 8∼9              // segment "Beverly Hills"
3 3∼4∼5∼6,8∼9      // segment set "Clampett let him go...Beverly Hills"
```

Listing 6: Segment Encoding

Segments are important when the text changes: If a new token is inserted into a segment, then the segment will include the new token going forward. If, on the other hand, a new token is inserted between two segments making up a segment set, the new token will not be included.

It is important to note that segments can be comprised not only of tokens, but of annotations in other layers of representation. A segment or segment set, however, may only be comprised of annotations from a single layer. Because token segments and segment sets are so common, though, in later BNF specifications of annotation `PCDATA` syntax, a token segment will be denoted by `TSEGMENT` and a token segment set by `TSEGMENTSET` without further elaboration.

3.6 Sentences

The Sentence annotation layer indicates the boundaries of sentences in the text. They are encoded as a single `TSEGMENT`, as shown in Listing 7.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
This is a story about a man named Jed.
</desc>
  </rep>
...
  <rep id="edu.mit.parsing.sentence">
    <desc id="26" len="38" off="0">16~17~18~19~20~21~22~23~24~25</desc>
  </rep>
</story>
```

Listing 7: Example Sentence Encoding. Token `id` numbers are taken from Listing 4.

3.7 Parts of Speech

The part of speech layer assigns a part of speech tag to either a token annotation or a MWE annotation. The only tagset currently allowed in the Penn Treebank tagset.

Symbol		Expression
PCDATA	::=	ID "␣" POSTAG
TID	::=	id of a token or MWE annotation
POSTAG	::=	"CC" | "CD" | "DT" | "EX" | "FW" | "IN" | "JJ" | "JJR" | "JJS" |"LS" | "MD" | "NN" | "NNS" | "NNP" | "NNPS" | "PDT" | "POS" | "PRP" | "PRP$" \| "RB" \| "RBR" \| "RBS" \| "RP" \| "SYM"\| "TO" \| "UH" \| "VB" \| "VBD" \| "VBG" \| "VBP" \| "VBZ" \| "WDT" \| "WP" \| "WP$" | "WRB" | "#" | "$" | "." | ":" | "(" | ")" | """ | "‘" | "’‘" | "’" | "’’"

Table 2: BNF for Part of Speech Representation. The two fields are space-delimited, as indicated by the ␣ symbol.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
This is a story about a man named Jed.
</desc>
  </rep>
...
  <rep id="edu.mit.parsing.pos">
    <desc id="27" len="4" off="0">16 DT</desc>
    <desc id="28" len="2" off="5">17 VBZ</desc>
    <desc id="29" len="1" off="8">18 DT</desc>
    <desc id="30" len="5" off="10">19 NN</desc>
    <desc id="31" len="5" off="16">20 IN</desc>
    <desc id="32" len="1" off="22">21 DT</desc>
    <desc id="33" len="3" off="24">22 NN</desc>
    <desc id="34" len="5" off="28">23 VBN</desc>
    <desc id="35" len="3" off="34">24 NNP</desc>
    <desc id="36" len="1" off="37">25 .</desc>
  </rep>
</story>
```

Listing 8: Example Part of Speech Encoding. Token `id` numbers are taken from Listing 4.

3.8 Stems / Lemmas

The Stem layer (also known as the Lemma layer) encodes a root form for a token or multi-word expression. If the stem is a stem of a multi-word expression and includes a space, the spaces are replaced with underscore characters "_".

Symbol		Expression
PCDATA	::=	ID "␣" ROOTFORM
ID	::=	id of a token or MWE annotation
ROOTFORM	::=	PCDATA, no whitespace allowed

Table 3: BNF for Stem Representation. The two fields are space-delimited, as indicated by the ␣ symbol.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
This is a story about a man named Jed.
</desc>
  </rep>
...
  <rep id="edu.mit.parsing.stem">
    <desc id="37" len="2" off="5">28 be</desc>
    <desc id="38" len="5" off="28">34 name</desc>
  </rep>
</story>
```

Listing 9: Example Stem Encoding. Token `id` numbers are taken from Listing 4.

3.9 Parse Trees

The Parse Tree layer encodes a Penn-Treebank-Style CFG parse tree for a set of tokens. The format is a parenthesis-encoded phrase structure parse tree, in the Penn Treebank format, with no linebreaks. Each leaf token string is appended with an underscore followed by the `id` number of the token annotation for that leaf.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
This is a story about a man named Jed.
</desc>
  </rep>
...
  <rep id="edu.mit.parsing.parse">
    <desc id="40" len="38" off="0">(ROOT (S (NP (DT This_16)) (VP (VBZ
       is_17) (NP (NP (DT a_18) (NN story_19)) (PP (IN about_20) (NP (NP (
       DT a_21) (NN man_22)) (VP (VBN named_23) (NP (NNP Jed_24)))))))) (.
       ._25)))</desc>
  </rep>
</story>
```

Listing 10: Example Parse Tree Encoding. Token `id` numbers are taken from Listing 4.

3.10 Wordnet Senses

The Wordnet Sense layer encodes Wordnet sense for each token or multi-word expression. The Wordnet Sense layer takes a single parameter indicating the version of wordnet in use (which is indicated by a unique identifier).

The `PCDATA`string for a word sense annotation is a comma-delimited list of fields, as shown in the following BNF.

Symbol		Expression
PCDATA	::=	WID "," EVIDENCE "," TID "," PID "," SID
WID	::=	"WID-" OFFSET "-" POS "-" WNUM "-" LEMMA
OFFSET	::=	Eight-digit, zero-padded synset offset
POS	::=	"V" | "N" | "R" | "J"
WNUM	::=	Two-digit, zero-padded word number
LEMMA	::=	Lemma of the assigned word
EVIDENCE	::=	"" | (floating point number) | (TYPE, ":", CONTENT)
TYPE	::=	"USER" | "MONOSEM" | "NUMERIC" | "CONTEXT"
CONTENT	::=	Arbitrary string, not containing ',' or ':', whose format depends on the evidence type
TID	::=	id number of the associated token or MWE annotation
PID	::=	id number of the associated part of speech annotation, if any
SID	::=	id number of the associated stem annotation, if any

Table 4: BNF for Wordnet Sense Representation.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <story>
3    <rep id="edu.mit.story.char">
4  This is a story about a man named Jed.
5  </desc>
6    </rep>
7  ...
8    <rep id="edu.mit.wordnet.sense">
9      <param wordnet="edu.princeton.wordnet.30"/>
10     <desc id="68" len="2" off="5">WID-02604760-V-01-be,,17,28,67</desc>
11     <desc id="41" len="5" off="10">WID-06369829-N-01-story,,19,30,</desc>
12     <desc id="57" len="3" off="24">WID-10287213-N-01-man,,22,33,</desc>
13     <desc id="59" len="5" off="28">WID-01028748-V-01-name,,23,34,38</desc>
14     <desc id="69" len="3" off="34">WID-00007846-N-01-person,,24,35,</desc>
15   </rep>
16 </story>
```

Listing 11: Example Wordnet Sense Encoding. Token `id` numbers are taken from Listing 4. Stem `id` numbers are taken from Listing 9. Part of Speech `id` numbers are taken from Listing 8.

3.11 Semantic Roles

This layer encodes semantic roles for verbs in sentences. The format of the `PCDATA` is exactly the same as that defined by the PropBank annotation file format, for individual semantic roles. As a general recap, the fields mean the following:

1. Leaf number of the verb (e.g., `1`)
2. the annotator; this defaults to `user` in the story workbench file format
3. the frameset id (e.g., `be.01`)
4. five flags indicating syntactic features, where '-' means no value (e.g., `----a`)
5. all remaining fields are arguments, with a node-span list, argument label, and argument feature

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <story>
3    <rep id="edu.mit.story.char">
4  This is a story about a man named Jed.
5  </desc>
6    </rep>
7  ...
8    <rep id="edu.mit.semantics.semroles">
9      <desc id="70" len="37" off="0">1 user be.01 ----a 0:1-ARG1- 2:2-ARG2-<
           /desc>
10     <desc id="71" len="15" off="22">7 user name.01 ----a 5:1-ARG0- 8:1-
           ARG1-</desc>
11   </rep>
12 </story>
```

Listing 12: Semantic Role Representation

3.12 Referring Expressions

The Referring Expression layer indicates where there are phrases that "refer" to things in the world, or participate in co-reference relationships. Each referring expression is encoded as a `TSEGMENTSET`.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
    <desc id="0" len="47" off="0">
John loved his horse.  It was a white stallion.
</desc>
  </rep>
...
  <rep id="edu.mit.parsing.token">
    <desc id="2" len="4" off="0">John</desc>
    <desc id="3" len="5" off="5">loved</desc>
    <desc id="4" len="3" off="11">his</desc>
    <desc id="5" len="5" off="15">horse</desc>
    <desc id="6" len="1" off="20">.</desc>
    <desc id="7" len="2" off="23">It</desc>
    <desc id="8" len="3" off="26">was</desc>
    <desc id="9" len="1" off="30">a</desc>
    <desc id="10" len="5" off="32">white</desc>
    <desc id="11" len="8" off="38">stallion</desc>
    <desc id="12" len="1" off="46">.</desc>
  </rep>
  <rep id="edu.mit.discourse.rep.refexp">
    <factory id="edu.mit.discourse.rep.refexp.factory.null"/>
    <desc id="13" len="4" off="0">2</desc>
    <desc id="15" len="3" off="11">4</desc>
    <desc id="16" len="9" off="11">4∼5</desc>
    <desc id="18" len="2" off="23">7</desc>
  </rep>
</story>
```

Listing 13: Example Referring Expression Encoding.

3.13 Coreference Groups

Coreference groups gather together sets of referring expressions that refer to the same referent, along with an annotator-supplied label, which may be anything,

Symbol		Expression
PCDATA	::=	NAME "|" REFEXPSET
NAME	::=	PCDATA
REFEXPSET	::=	Comma-delimited list of Referring Expression annotation id numbers

Table 5: BNF for Coreference Representation.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
    <desc id="0" len="47" off="0">
John loved his horse.  It was a white stallion.
</desc>
  </rep>
...
  <rep id="edu.mit.discourse.rep.coref">
    <desc id="14" len="14" off="0">John|13,15</desc>
    <desc id="17" len="14" off="11">Stallion|16,18</desc>
  </rep>
</story>
```

Listing 14: Example Coreference Encoding. Referring Expression `id` numbers are taken from Listing 13.

3.14 Referent Properties

Referent properties mark permanent, unchanging properties of referents. They consist of the following fields, pipe-delimited:

1. `id` number of the referring expression to which the property attaches
2. a tag drawn from the list of tags set out by the annotation guide
3. a token segment set capturing the tokens that express the property

Symbol		Expression
PCDATA	::=	REFEXPID "|" TAG "|" TSEGMENTSET
REFEXPID	::=	id number of target referring expression
TAG	::=	DESCRIPTIVE | PHYSICAL | MATERIAL | LOCATION | PERSONALITY | NAME_OR_TITLE | ORDINAL | AMOUNT_COUNTABLE | AMOUNT_MASS | QUANTIFICATION | ORIGIN | LOCATION | MATERIAL | WHOLE | UNKNOWN

Table 6: BNF for Referent Properties Representation.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
    <desc id="0" len="47" off="0">
John loved his horse.  It was a white stallion.
</desc>
  </rep>
...
  <rep id="edu.mit.discourse.rep.refprop" ver="0.2.0">
    <desc id="22" len="9" off="11">16|CLASS|5</desc>
    <desc id="20" len="14" off="23">18|PHYSICAL|10</desc>
    <desc id="21" len="23" off="23">18|CLASS|11</desc>
  </rep>
</story>
```

Listing 15: Referent Properties Representation

3.15 Events

The Event Representation captures the TimeML annotation of events in the text. It is encoded as eleven pipe-delimited fields, as follows:

Symbol		Expression
PCDATA	::=	TYPE "|" HEAD "|" EXTENT "|" POS "|" TENSE "|" ASPECT "|" POLARITY "|" PSIGNAL "|" CARDINALITY "|" CSIGNAL "|" MSIGNAL
TYPE	::=	"UNSPECIFIED" | "OCCURRENCE" | "STATE" | "PERCEPTION" | "REPORTING" | "ASPECTUAL" | "I_ACTION" | "I_STATE" | "UNKNOWN
HEAD	::=	TSEGMENTSET of the head of the event
EXTENT	::=	TSEGMENTSET of the full extent of the event
POS	::=	"VERB" | "NOUN" | "ADJECTIVE" | "ADVERB" | "PREPOSITION" | "OTHER" | "UNSPECIFIED"
TENSE	::=	"NONE" | "UNSPECIFIED" | "PAST" | "PRESENT" | "FUTURE" | "INFINITIVE" | "PRESPART" | "PASTPART"
ASPECT	::=	"NONE" | "UNSPECIFIED" | "PERFECT" | "PROGRESSIVE" | "BOTH"
POLARITY	::=	"true" | "false"
PSIGNAL	::=	TSEGMENTSET of the tokens indicating the polarity
CARDINALITY	::=	a positive integer
CSIGNAL	::=	TSEGMENTSET of the tokens indicating the cardinality
MSIGNAL	::=	TSEGMENTSET of the tokens indicating the modality

Table 7: BNF for Event Representation.

An example encoding of events is shown in Listing 16.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
    <desc id="0" len="58" off="0">
John arrived at 2:00 PM.  He then tried to find his hotel.
</desc>
  </rep>
...
  <rep id="edu.mit.parsing.token">
    <desc id="2" len="4" off="0">John</desc>
    <desc id="3" len="7" off="5">arrived</desc>
    <desc id="4" len="2" off="13">at</desc>
    <desc id="5" len="4" off="16">2:00</desc>
    <desc id="6" len="2" off="21">PM</desc>
    <desc id="7" len="1" off="23">.</desc>
    <desc id="8" len="2" off="26">He</desc>
    <desc id="9" len="4" off="29">then</desc>
    <desc id="10" len="5" off="34">tried</desc>
    <desc id="11" len="2" off="40">to</desc>
    <desc id="12" len="4" off="43">find</desc>
    <desc id="13" len="3" off="48">his</desc>
    <desc id="14" len="5" off="52">hotel</desc>
    <desc id="15" len="1" off="57">.</desc>
  </rep>
  <rep id="edu.mit.semantics.rep.event">
    <desc id="16" len="7" off="5">OCCURRENCE|3|3|VERB|PAST|NONE|true||1||<
       /desc>
    <desc id="17" len="5" off="34">I_ACTION|10|10|VERB|PAST|NONE|true||1||
       </desc>
    <desc id="18" len="7" off="40">OCCURRENCE|12|11~12|VERB|INFINITIVE|
       NONE|true||1||</desc>
  </rep>
</story>
```

Listing 16: Example Event Encoding.

3.16 Time Expressions

The Time Expression Representation captures the TimeML annotation of Timex3 (temporal expressions) in the text. It is encoded as six pipe-delimited fields, as follows:

Symbol		Expression
PCDATA	::=	TYPE "|" MOD "|" EXTENT "|" FUNCTION "|" QUANT "|" FREQ
TYPE	::=	"DATE" | "TIME" | "DURATION" | "SET" | "UNKNOWN" | "UNSPECIFIED"
MOD	::=	"NONE" | "BEFORE" | "AFTER" | "ON_OR_BEFORE" | "ON_OR_AFTER" | "LESS_THAN" | "MORE_THAN" | "EQUAL_OR_LESS" | "EQUAL_OR_MORE" | "START" | "MID" | "END" | "APPROX" | "UNKNOWN" | "UNSPECIFIED"
EXTENT	::=	TSEGMENTSET of the time expression extent
FUNCTION	::=	"NONE" | "CREATION_TIME" | "EXPIRATION_TIME" | "MODIFICATION_TIME" | "PUBLICATION_TIME" | "RELEASE_TIME" | "RECEPTION_TIME" | "UNKNOWN" | "UNSPECIFIED"
QUANT	::=	TSEGMENTSET of the quantifier extent
FREQ	::=	TSEGMENTSET of the frequency extent

Table 8: BNF for Time Expression (Timex3) Representation.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
    <desc id="0" len="58" off="0">
John arrived at 2:00 PM.  He then tried to find his hotel.
</desc>
  </rep>
...
  <rep id="edu.mit.semantics.timex3">
    <desc id="19" len="7" off="16">TIME|NONE|5~6|NONE||</desc>
  </rep>
</story>
```

Listing 17: Example Timex3 Encoding. Token `id` numbers are taken from Listing 16.

3.17 Temporal Links

The Time Link Representation captures the TimeML annotation of temporal links in the text. It is encoded as five pipe-delimited fields, where the second field depends on the first, as follows:

Symbol		Expression
PCDATA	::=	CLASS "|" ARG1 "|" ARG2 "|" SIGNAL
CLASS	::=	("TEMPORAL" "|" TSUBTYPE) | ("SUBORDINATING" "|" SSUBTYPE) | ("ASPECTUAL" "|" ASUBTYPE)
TSUBTYPE	::=	"BEFORE" | "AFTER" | "INCLUDES" | "IS_INCLUDED" | "DURING" | "DURING_INV" | "SIMULTANEOUS" | "IAFTER" | "IBEFORE" | "IDENTITY" | "BEGINS" | "ENDS" | "BEGUN_BY" | "ENDED_BY" | "UNSPECIFIED" | "UNKNOWN"
SSUBTYPE	::=	"MODAL" | "EVIDENTIAL" | "NEG_EVIDENTIAL" | "FACTIVE" | "COUNTER_FACTIVE" | "CONDITIONAL" | "UNKNOWN" | "UNSPECIFIED"
ASUBTYPE	::=	"INITIATES" | "CULMINATES" | "TERMINATES" | "CONTINUES" | "REINITIATES" | "UNKNOWN" | "UNSPECIFIED"
ARG1	::=	id number of an Event or Time expression annotation
ARG2	::=	id number of an Event or Time expression annotation
SIGNAL	::=	TSEGMENTSET covering the signal of the link

Table 9: BNF for Time Expression (Timex3) Representation.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <story>
3    <rep id="edu.mit.story.char">
4      <desc id="0" len="58" off="0">
5  John arrived at 2:00 PM.  He then tried to find his hotel.
6  </desc>
7    </rep>
8  ...
9    <rep id="edu.mit.semantics.rep.timelink">
10     <desc id="20" len="18" off="5">TEMPORAL|SIMULTANEOUS|16|19|4</desc>
11     <desc id="21" len="34" off="5">TEMPORAL|AFTER|16|17|9</desc>
12     <desc id="22" len="13" off="34">SUBORDINATING|FACTIVE|17|18|</desc>
13   </rep>
14 </story>
```

Listing 18: Representation. Token and Event `id` numbers are taken from Listing 16. Time Expression `id` numbers are taken from Listing 17.

3.18 Tags

The Tag representation allows for annotation of arbitrary tag sets over arbitrary spans of text. It consists of two fields:

Symbol		Expression
PCDATA	::=	ID "|" EXTENT
ID	::=	a unique string identifying the tag
EXTENT	::=	TSEGMENTSET covering the extent of the tag

Table 10: BNF for the Tag Representation.

The allowed tag IDs, and their meaning, is defined in the file pointed to by the `tagsetPath` parameter. An example tagset file is shown in Listing 20.

```
<?xml version="1.0" encoding="UTF-8"?>
<story>
  <rep id="edu.mit.story.char">
    <desc id="0" len="38" off="0">
This is a story about a man named Jed.
</desc>
  </rep>
...
  <rep id="edu.mit.parsing.tag" ver="1.1.0">
    <param tagsetPath="test.tagset"/>
    <factory id="edu.mit.parsing.tag.factory.null"/>
    <desc id="26" len="7" off="8">NOUN|18∼19</desc>
    <desc id="27" len="15" off="22">NOUN|21∼22∼23∼24</desc>
  </rep>
</story>
```

Listing 19: Example Tag Encoding. Token `id` numbers are taken from Listing 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<TAGSET version="1.0"
        id="edu.mit.parsing.core.tagset.test"
        name="Test Tagset"
        desc="A tagset for testing purposes"
        target="edu.mit.parsing.token"
        scope="segment">
  <TAG id="NOUN" name="Noun"     desc="Nouns or noun phrases"/>
  <TAG id="VERB" name="Verb"     desc="Verbs or verb phrases"/>
  <TAG id="ADV" name="Adverb"    desc="Adverbs or adverbial phrases/>
  <TAG id="ADJ" name="Adjective" desc="Adjectives or adjectival phrases"/>
</TAGSET>
```

Listing 20: Example Tag Set Definition File.

4 Metadata Formats

Metadata annotations are annotations on top of normal annotations. Each piece of metadata takes a `descID` and `repID` which indicates the specific annotation to which the metadata attaches. The annotation need not actually exist in the document (it may have been deleted, but the metadata retained).

4.1 Checks

Checks indicate that the annotator has indicated a particular problem or warning has been "checked". The `PCDATA`of this metadata consists of a problem identifier that has been marked as "checked" on the particular annotation.

4.2 Marks

A mark is a UI construct that allows a particular annotation to be pointed out for later evaluation. The `PCDATA`for the mark metadata is empty.

4.3 Merge Tags

A merge tag is a comma-delimited list of annotator or document identifier strings, indicating the source of the annotation. This metadata type is used when two documents are merged together into one.

4.4 Notes

A note is an arbitrary piece of text attached to an annotation. The `PCDATA`is the actual note.

4.5 Origin

An origin indicates who originally created the annotation. If it was the annotator, the `PCDATA`value is `user`. If it was a factory, then the `PCDATA`is the identifier of the factory.

4.6 Timing

Timing metadata indicates when an annotation was modified. It consists of three pipe-delimited fields:

Symbol		Expression
PCDATA	::=	START "|" END "|" SOURCE
START	::=	a time string, indicating the creation time or start time of editing, e.g., "2012-02-22+17:20:44.524-0500"
END	::=	the end time of editing, may be empty
SOURCE	::=	"user" if it was a change initiated by the annotator; otherwise empty

Table 11: BNF for Timing Metadata.

2012-02-22+17:20:44.524-0500