MMAX2 Annotation Schemes (draft)

© Christoph Müller
EML Research gGmbH
http://mmax.eml-research.de

9th February 2005

Contents

1 About this Document

This document describes MMAX2 Annotation Schemes and how these can be used to define attributes, values and relations of markables. Within the MMAX2 framework, annotation schemes are of crucial importance since they specify the 'vocabulary' available for describing phenomena for manual annotation. **Note:** This document contains information required for defining and customizing annotation schemes by hand, i.e. by manually modifying the XML files in which they are specified. While this is easy and feasible for shorter and simpler schemes, it can get difficult and awkward for longer ones, especially when more complex hierarchical relations are used. Therefore, a graphical user interface for annotation scheme definition will be added in some later version.

Note: The annotation scheme mechanisms have been overhauled in version 1.0 beta 4. Annotation scheme definition ist now simpler and more straightforward, especially if *attribute dependencies* (cf. Section 4.1) are used. As a result, some annotation schemes defined for earlier versions might behave slightly differently. If you experience this and have problems adapting your schemes yourself, please do not hesitate to contact us at mmax@eml-research.de, and we will be happy to assist you.

2 Attributes and Relations

Each annotation scheme is basically about attributes and relations. It is in terms of these two categories that an annotation scheme is specified. Each attribute and each relation has to be of a certain type. Currently, MMAX2 supports the following attribute types:

- FREETEXT:
  An attribute of this type can accept as its value any string.

- NOMINAL_LIST:
  An attribute of this type can accept as its value one of a closed set of values defined for it in the annotation scheme. The term LIST refers to the fact that attributes of this type are rendered as drop-down lists in the MMAX2 Attribute Window. In all other respects, attributes of this type are identical to attributes of the type described next.

- NOMINAL_BUTTON:
  An attribute of this type can accept as its value one of a closed set of values defined for it in the annotation scheme. The term BUTTON refers to the fact that attributes of this type are rendered as a sequence of radio buttons in the MMAX2 Attribute Window. In all other respects, attributes of this type are identical to attributes of the type described above.

The following is a list of all currently supported relation types:

- MARKABLE_SET:
  A relation of this type can accept as its value an ID identifying a set of one or more markables which have the same ID in the same relation. In other words, a relation of this type can be used to *group* two or more markables together, forming a set. The value of this relation is never set explicitly by the user, but only by means of adding and removing markables from sets in the MMAX2 GUI.

- MARKABLE_POINTER:
  A relation of this type can accept as its value one or more IDs of other markables. Thus, it can be used to model a *pointing* relation from one markable to one or more other markables. As with MARKABLE_SET relation, the value of this relation is never set explicitly by the user, but only by means of adding and removing pointers to markables in the MMAX2 GUI.

Both attributes and relations are defined in a so-called annotation scheme file, which contains XML elements specifying attributes, their type, and, if applicable, their possible values. This is done in a sequence of <attribute> and embedded <value> tags. The XML tags look slightly different, depending on whether they define an attribute or a relation. Schematically, the XML tags look as follows.

- FREETEXT attributes:
  <attribute id=ATT_ID name=ATT_NAME type="freetext" text=ATT_DESC>
  <value id=ATT_VAL_ID name=ATT_NAME/ >
  </attribute>
  The value of the name attribute is used both for display in the attribute window, and as the name of the XML tag on a markable. Text found in the optional text attribute is displayed in the attribute window as a tool tip when the mouse rests over the corresponding name label. FREETEXT attributes must have exactly one <value/> tag, and this tag's name attribute must be identical to the name value of the enclosing <attribute> tag.

- NOMINAL attributes:
  <attribute id=ATT_ID name=ATT_NAME type="nominal_button|nominal_list"
  text=ATT_DESC>
  <value id=ATT_VAL_ID name=VAL_NAME text=VAL_DESC next=DEP_ATTS/>
  ...
  </attribute>
  The value of the name attribute is used both for display in the attribute window, and as the name of the XML tag on a markable. Text found in the optional text attribute on either the <attribute> or the <value> tag is displayed in the attribute window as a tool tip when the mouse rests over the corresponding name label. NOMINAL attributes must have one <value/> tag for every value defined for them. Each value can have an optional next attribute which can contain a (list of) IDs of <attribute> tags *dependent* on them. See Section 4 for details. Note that the difference between NOMINAL_LIST and NOMINAL_BUTTON only relates to the way they are displayed in the attribute window. A NOMINAL_LIST attribute can be turned into a NOMINAL_BUTTON one, and vice versa, by simply changing the value of the type attribute. It can be useful, e.g. to use the BUTTON type for annotation,[1] since it allows for quicker value selection, and change it to LIST for browsing the completed annotation, since this display form takes less space.

- MARKABLE_SET:
  <attribute id=ATT_ID name=REL_NAME type="markable_set" text=REL_DESC
  style="straight|lcurve|rcurve|xcurve"
  color=LINE_COLOR
  width=LINE_WIDTH
  add_to_markableset_text=ITEM_TEXT
  remove_from_markableset_text=ITEM_TEXT
  adopt_into_markableset_text=ITEM_TEXT
  merge_into_markableset_text=ITEM_TEXT>
  <value id=ATT_VAL_ID name=REL_NAME/>
  </attribute>
  The value of the name attribute is used both for display in the attribute window, and as the name of the XML tag on a markable. Text found in the optional text attribute is displayed in the attribute window as a tool tip when the mouse rests over the corresponding name label. The optional style attribute (default value = straight) can be used to control the form of the line used to visualize a markable set if one of its members is selected in the MMAX2 main window. The optional color attribute (default value = black) can specify the color of this line. Possible pre-defined values are: black, blue, cyan, lightGray, gray, darkGray, green, magenta, orange, pink, red, white, yellow. Values can also be specified as triples of RGB-values, either with decimal (three-place!) values prepended with 'd:' (d:rrrgggbbb) or with hexadecimal (two-place!) values prepended with 'x:' (x:rrggbb). The optional width attribute (default value = 2) can be used to control the width of the line. Note that XML requires that this attribute's value also be enclosed in quotation marks, even though it is a number! The optional *_text attributes can be used to customize the appearance of the popup menu used for manipulating markable sets in the MMAX2 main window. If left out, default values will be used. MARKABLE_SET attributes must have exactly one <value/> tag, and this tag's name attribute must be identical to the name value of the enclosing <attribute> tag.

[1] Though only if the list of possible values is not too long!

- MARKABLE_POINTER:
  <attribute id=ATT_ID name=REL_NAME type="markable_pointer" text=REL_DESC
  style="straight|lcurve|rcurve|xcurve"
  color=LINE_COLOR
  width=LINE_WIDTH
  max_size=MAX_TARGET_NUMBER
  target_domain=TARGET_DOMAINS
  point_to_markable_text=ITEM_TEXT
  remove_pointer_to_markable_text=ITEM_TEXT/>
  <value id=ATT_VAL_ID1 name="not_set" next=DEP_ATTS/>
  <value id=ATT_VAL_ID2 name="set" next=DEP_ATTS/>
  </attribute>
  The value of the `name` attribute is used both for display in the attribute window, and as the name of the XML tag on a markable. Text found in the optional `text` attribute is displayed in the attribute window as a tool tip when the mouse rests over the corresponding name label. The optional `style` attribute (default value = straight) can be used to control the form of the line used to visualize pointing relations to other markables if the source markable is selected in the MMAX2 main window. The optional `color` attribute (default value = black) can specify the color of this line. Possible pre-defined values are: black, blue, cyan, lightGray, gray, darkGray, green, magenta, orange, pink, red, white, yellow. Values can also be specified as triples of RGB-values, either with decimal (three-place!) values prepended with 'd:' (d:rrrgggbbb) or with hexadecimal (two-place!) values prepended with 'x:' (x:rrggbb). The optional `width` attribute (default value = 2) can be used to control the width of the line. The optional `max_size` attribute (default value = infinity) can be used to limit the maximum number of markables a source markable may point to via this relation. Note that XML requires that the latter two attributes' values also be enclosed in quotation marks, even though they are numbers! Finally, the optional `target_domain` attribute (default value = all levels) can be used to restrict to markables on which levels a markable may point via this relation. The value of this attribute has to be specified as a comma-separated list **without any space characters** in between. The optional `*_text` attributes can be used to customize the appearance of the popup menu used for manipulating markable pointers in the MMAX2 main window. If left out, default values will be used. MARKABLE_POINTER attributes must have exactly two <`value/`> tags with the name values *not_set* and *set*, as shown above. Each of these can have an optional `next` attribute which can contain a (list of) IDs of <`attribute`> tags *dependent* on them. See Section 4 for details.

3 Simple Schemes

For some annotation tasks, rather simple annotation schemes can be sufficient. In the following, an annotation scheme is referred to as *simple* if it does not employ attribute dependencies, i.e. if none of the <`value`> tags defined in it make use of the `next` attribute.
Every MMAX2 annotation scheme must contain the top level <`annotationscheme`> tag. In addition, since the annotation scheme file is in XML format, it has to contain the <`?xml version="1.0"?`> tag as its first content. Thus, a *minimal* annotation scheme with no attributes looks like this:

```
<?xml version="1.0"?>
<annotationscheme>
```

```
</annotationscheme>
```

Each markable level must have access to an annotation scheme file with *at least* the above content. For levels which do not contain any attributes or relations, nothing more has to be added. However, if markables on a particular level are supposed to have attributes and relations (the normal case), these have to be defined in the annotation scheme. Otherwise, selecting a markable in the MMAX2 gui will trigger an error message stating that an **illegal** attribute has been found on the markable.[2] The same is true if the *attribute* itself is defined in the annotation scheme, but the *value* found on the particular markable is not.
Attributes are defined in the annotation scheme by simply adding the respective <attribute> elements as described in Section 2. There are no constraints on the order in which <attribute> elements can appear. In the attribute window, the attributes will be displayed in the order in which they appear in the annotation scheme file.
Each attribute also has a *default* value. For FREETEXT attributes, the default value is the empty string. For MARKABLE_SET and MARKABLE_POINTER relations, the default value also is the empty string, meaning *no relation assigned*. For NOMINAL attributes, the *first* value defined in the annotation scheme will be used as the default value. **Tip:** It can be advisable sometimes to specify some special value (like e.g. *none*) as the default value for a NOMINAL attribute, in order to signal that for a given markable the value for this attribute has not yet been assigned by the annotator.

4 Complex Schemes

4.1 Overview

In contrast to simple schemes, *complex* schemes are those that employ dependencies between attributes. Dependencies are specified by means of the next attribute on <value/> tags in the annotation scheme. In a nut shell, one or more attributes are dependent on some other attribute A if they are available only if A has a particular value, i.e. the value that has the dependent attribute(s) in its next attribute. It follows from this that only those attributes with more than one <value/> tag, i.e. all except FREETEXT and MARKABLE_SET attributes, can have dependent attributes. *Independent* attributes, on the other hand, are those whose IDs do not appear in any <value/> tag's next attribute, i.e. those that always apply regardless of the current values of other attributes. It follows from this that each annotation scheme must have *at least one* independent attribute. An attribute from which other attributes are dependent is called a *branching* attribute.
Attribute dependencies are resolved by **recursively expanding** a list of attributes. Given an initial list of *n>=1* attributes / relations, this list is processed from top to bottom, the attribute at index X being the *current attribute*. For each current attribute or relation, it is then determined whether it *currently* has some valid dependent attributes. This is done by checking whether the <value/> tag corresponding to the attribute's current value has some non-empty next attribute, specifying IDs of dependent attributes. If the current <value/> tag does have such a list, these dependent attributes are retrieved. This produces another list of attributes, ordered in the sequence in which they are referenced in the next attribute. This list is then **appended** to the current attribute list at index X+1, i.e. **directly after** the current attribute, thus increasing the overall length of the list. In the next iteration, the whole process is continued with X+1. If the attribute at index X did have some dependent attributes, the new current atribute at X+1 will be the first

[2] This can be suppressed by deselecting the *Warn on extra attributes* option in the MMAX2 Attribute Window.

of these. Note that dependent attributes can themselves have attributes dependent on them, so that *multiple dependencies* can be expressed.
One attribute can be dependent on several other attributes as long as these are **mutually exclusive**. This means that one attribute can appear in the `next` attribute of e.g. the <`value/`> tag of attribute A and B, as long as A and B do never apply simultaneously. A necessary (but not sufficient!) prerequisite for mutual exclusion is that neither A nor B be independent attributes.
Individual values for one attribute are per definition mutually exclusive. Therefore, it is possible for one attribute to appear in the `next` attribute of more than one <`value/`> tag of a given attribute.

4.2 Markable Creation

When a new markable is created, all *independent* attributes will be applied to it with their respective *default* values (cf. Section 3). As in simple schemes, independent attributes in complex schemes can be added to the annotation scheme in any order, and they will be displayed in the MMAX2 Attribute Window in the order in which they appear in the annotation scheme. Note that default values can also have `next` attributes, i.e. there can be attributes that depend on some other attribute having the default value. Therefore, the initial list of independent attributes is recursively expanded as described in Section 4.1 above. When no more attributes can be added recursively, the resulting attributes are applied to the newly created markable. Note that the newly created markable need not be selected in order to have the attributes applied to it.

4.3 Markable Selection

When an existing markable is selected via left-click in the MMAX2 GUI, its attributes are validated against the annotation scheme defined for the level the markable belongs to. This is done by first retrieving the list of all independent attributes, as described above. This list is iterated from top to bottom, the attribute at index X being the *current attribute*.
If an attribute of the same name as X is found on the currently selected markable, it is tried to set it to that value. This will fail if the value found is not defined as a possible value of attribute X, and a message will be displayed to the user.[3] In any case, the attribute found on the markable will be marked internally as *processed*. Finally, it is checked whether there are dependent attributes. If so, these are retrieved and added to the list of attributes as described in Section 4.1 above.
If, on the other hand, no attribute with the same name as attribute X is found on the currently selected markable, attribute X is set to default. This will never fail since the default value will always be defined in the annotation scheme. The attribute found on the markable will also be marked internally as *processed*. Then, a check for dependent attributes will be peformed, and any attributes found will be added to the list as described above.
After all independent attributes have been recursively mapped to the currently selected markable, all attributes found on that markable should be marked as *processed*. If this is not the case, the markable contains one or more undefined *extra* attribute(s), and a corresponding message will be displayed to the user. Note that this message can be suppressed by unchecking the option *Warn on extra attributes* in the MMAX2 Attribute Window. Note also that there is one such option for each defined markable level, so different settings can be applied to different levels.

[3] This can happen if markable files have been created externally and not via the MMAX2 GUI.

4.4 Markable Attribute Modification

A good deal of manual annotation work consists in modifying markable attributes via the MMAX2 Attribute Window. In order to modify a markable's attributes, it has to be selected by left-click.[4] The Attribute Window is then set to display the attributes retrieved by means of the process described in Section 4.3 above. Branching attributes, i.e. ones which have dependent attributes, are displayed with a <> before their name.

4.4.1 NOMINAL Attributes

NOMINAL attributes can be modified by selecting a different than the currently selected value for them. Depending on the subtype (LIST vs. BUTTON), this is done by selecting a different entry in the drop down list, or clicking a different radio button. If the associated attribute is either not branching at all, or if the old and new value do not have `next` attributes, nothing else will happen. Otherwise, the display will change depending on what the `next` attributes on the old and the new value were.
If the old value had some dependent attributes which now do not apply any more (since the old value is deselected), these dependent attributes[5] are removed from the display. Internally, the attributes and their respective values are stored temporarily, since some or all of them might also be dependent on the *new* value. Note that not necessarily *all* attributes below the modified one are removed, but only those that are (directly or indirectly) dependent on it.
If the new value has some dependent attributes, these are retrieved in the order in which they are referenced in the `next` attribute of the new value. It is possible that one or more attributes are removed and added at the same time, i.e. in cases where one or more attributes are in the `next` attribute of both the old and new value. In this case, as many values as possible from the old attributes are transferred to the new ones. The resulting list of new attributes is then recursively expanded as described in Section 4.1 above. In the process, it is also tried to map as many remaining old attributes as possible. If the list is completely expanded, the attributes contained in it are added to the display, thus reflecting the new valid attributes for the currently selected markable. Note that unless the *Auto-apply* option is selected, the new attributes will only be applied to the markable after the *Apply* button has been clicked. As long as this button has not been clicked, any modifications to the attribute window can be undone by clicking the *Undo* button. Note also that in order for the modifications to be permanent, the markable level containing the modified markables has to be saved.

4.4.2 MARKABLE_POINTER Relations

Relations of type MARKABLE_POINTER are the only other attribute type that supports dependent attributes. In contrast to NOMINAL attributes (cf. above), values of relations of type MARKABLE_POINTER are modified by means of mouse actions in the MMAX2 GUI. If a markable is selected via left-click, and if it has a relation of type MARKABLE_POINTER defined, the value for this relation can be modified by adding or removing pointers to other (compatible) markables via right-click. Relevant changes occur if the markable's value changes from *not_set* to *set* or vice versa. If the first pointer is added to the current markable, the value changes from *not_set* to *set*. If the last pointer is removed, it changes from *set* to *not_set*. As a result, attributes dependent on the old value are removed, and those dependent on the new one are added, just like with NOMINAL attributes (cf. Section 4.4.1 above).

[4] Relations are an exception to this, as they are modified by means of right-clicks in the MMAX2 GUI.
[5] And the ones in turn dependent on them!