The Kinemage File Format, v1.0

Ian W. Davis, Jane S. Richardson, David C. Richardson

2nd September 2003

Abstract

This document describes the syntax and semantics of the core feature-set of the kinemage file format. Since even the core features are extensive, the description has been broken into two logical halves.

The first part of the document describes the low-level syntax that is common to all kinemage formats, regardless of how many additional functionalities they incorporate. The descriptions are very precise, at the cost of being somewhat long and tedious. However, this level of detail is necessary for programmers who wish to interpret kinemage files reliably, and may be helpful to authors as well. This level is expected to be extremely stable and change very slowly.

The second part of the document describes the semantics and high-level syntax of kinemage files, giving descriptions of the most commonly used keywords and the various options and parameters that accompany them. This level of detail is important to both programmers and kinemage authors. While the core features are stable and will remain so, additional features may be added with some frequency.

The descriptions assume a similar division of labor in the implementation of computer programs that process kinemages. The low-level syntax is handled by a tokenizer, which can separate a stream of characters into meaningful atomic units. The semantics and high-level syntax are handled by a parser, which is responsible for understanding, *e.g.*, the relationships among graphics objects and their implied hierarchical organization.

Contents

Part I
Syntax

1 Characters in kinemage files

Kinemage files are plain text files encoded according to the ASCII standard[1], which defines 128 characters. Each character is stored in the lower 7 bits of a single byte. Only ASCII characters between 32 and 126 inclusive, plus 9 (horizontal tab), 10 (newline), 12 (formfeed), and 13 (carriage return) are legal characters in a kinemage file (numbers given are in decimal).

[1] See `http://www.asciitable.com/` for details.

A kinemage tokenizer may check for and report illegal characters, but is not required to. If the tokenizer does find illegal characters, they should not cause a fatal error, but should instead be treated as alphanumerics (see 1.2).

1.1 Whitespace

Whitespace characters are the space (32), horizontal tab (9), newline (10), formfeed (12), carriage return (13), and the comma (44). Commas are defined as whitespace to simplify treatment of a sequence of numbers, which is often writen out with commas as separators.

The kinemage format is whitespace insensitive: these characters carry no meaning and may be discarded at the tokenizer level. Where whitespace is called for, one or more whitespace characters may be used, and any sequence of continguous whitespace characters is treated as a single occurance of whitespace. However, there is one important semantic attribute conveyed by whitespace: the newline and carriage return characters impart the beginning-of-line (BOL) property to any token immediately following them. See 2.1 for details.

When using whitespace, keep in mind that kinemage files should be human-readable and human-editable. Line length should not exceed 80 characters, but superfluous line breaks should be avoided. Single spaces are the preferred form of whitespace within a line. These suggestions are merely matters of style, and a kinemage tokenizer must not rely on them being followed.

1.2 Alphanumerics

Alphanumeric characters are the uppercase letters A-Z, the lowercase letters a-z, and the digits 0-9. Note that kinemage files are case sensitive. Kinemage tokenizers must not convert or mangle the case of any tokens in a kinemage file, and tokens that differ only by case must still be considered distinct from one another.

1.3 Punctuation

All legal characters that are neither classified as whitespace nor as alphanumerics are regarded as punctuation. These characters have a variety of functions in the kinemage format. The following characters already have well-defined function and syntax associated with them:

```
@ ( ) - = + { } " ' < . >
```

At the moment, no special significance has been attached to the following characters:

```
` ~ ! # $ % ^ & * _ [ ] \ | : ; / ?
```

However, a future version of the format may define meanings for them.

2 Tokens in kinemage files

Files in kinemage format can be thought of a sequences of tokens (meaningful), each separated from the others by zero or more whitespace characters (meaningless). Tokens are divided into two classes, quoted and unquoted. Quoted tokens have clear start and end signals, so they can occur with no intervening whitespace and still be separable. Unquoted tokens lack clear start and/or end signals. Thus, at least one whitespace character is *required* between two unquoted tokens in order to separate them from one another.

Theoretically, each token may be of any length, from one character (even zero characters, for quoted tokens) up to the largest string that will fit in memory. In practice, however, tokens should be fairly short; 20 characters or less is a reasonable guideline. No token should exceed 256 characters in length, and more stringent restrictions on length may be imposed on some tokens by the higher-level syntax.

The names given to token types below reflect their usual function in a kinemage file, but they are not restricted to that function. For example, an identifier usually names some object, but it can also enclose a command line, a file name, and so on.

2.1 Beginning-of-line

Beginning-of-line (BOL) is a property of certain tokens that may influence their interpretation by the parser. For instance, for a token to be recognized as a keyword (see 2.3.1), it must occur at the beginning of a line. A token is considered BOL under any of the following conditions:

- The first character of the token is the first character in the file
- The first character of the token is immediately preceded by a newline
- The first character of the token is immediately preceded by a carriage return

2.2 Quoted tokens

Quoted tokens all have explicit markers for the beginning and end of the token. This simplifies the parsing of these tokens, and enables one to classify the type of token present after parsing the first character of it. However, care must be taken to close every token that is opened. To aid authors in discovering such errors in their kinemages, it is recommended that kinemage tokenizers report a non-fatal error when they encounter the end of the file before closing an open quoted token.

2.2.1 Identifiers

Identifiers are strings quoted by curly braces, like this:

```
{an identifier}
```

An identifier token begins when an opening curly brace is encountered outside of any other quoted token (but possibly "inside", *i.e.*, immediately following, an unquoted token). It terminates as soon as the number of closing curly braces encountered in the course of parsing this token equals the number of opening curly braces encountered. That is, curly braces may be nested within an identifier, but only as long as they are balanced. Otherwise, an identifier may contain any legal character for a kinemage file.

2.2.2 Comments

Comments are strings quoted by angle brackets, like this:

```
<a comment>
```

A comment token begins when an opening angle bracket is encountered outside of any other quoted token (but possibly "inside", *i.e.*, immediately following, an unquoted token). It terminates as soon as the number of closing angle brackets encountered in the course of parsing this token equals the number of opening angle brackets encountered. That is, angle brackets may be nested within a comment, but only as long as they are balanced. Otherwise, a comment may contain any legal character for a kinemage file.

2.2.3 Aspects

Aspects are strings quoted by parentheses, like this:

```
(an aspect)
```

An aspect token begins when an opening parenthesis is encountered outside of any other quoted token (but possibly "inside", *i.e.*, immediately following, an unquoted token). It terminates as soon as the number of closing parentheses encountered in the course of parsing this token equals the number of opening parentheses encountered. That is, parentheses may be nested within an aspect, but only as long as they are balanced. Otherwise, an aspect may contain any legal character for a kinemage file.

2.2.4 Single quoted strings (pointmasters)

Pointmasters are represented as strings deliminted by single quote marks, like this:

```
'abc'
```

A single quoted token begins when a single quote mark is encountered outside of any other quoted token (but possibly "inside", *i.e.*, immediately following, an unquoted token). It terminates as soon another single quote mark is encountered. That is, single quoted strings may not contain embedded single quotes, and no mechanism exists to escape this limitation. Otherwise, a single quoted string may contain any legal character for a kinemage file.

2.2.5 Double quoted strings

Double quoted strings are defined analogously to single quoted strings, like this:

```
"abc"
```

A double quoted token begins when a double quote mark is encountered outside of any other quoted token (but possibly "inside", *i.e.*, immediately following, an unquoted token). It terminates as soon another double quote mark is encountered. That is, double quoted strings may not contain embedded double quotes, and no mechanism exists to escape this limitation. Otherwise, a double quoted string may contain any legal character for a kinemage file.

At the moment, no function has been ascribed to double quoted strings in kinemage files. Until a meaning is defined, parsers should ignore them.

2.3 Unquoted tokens

Unquoted tokens are somewhat harder to parse than quoted tokens, because their start and end signals are less obvious. Also, the entire token may need to be parsed before one is able to decide what kind of token it is. However, rules for parsing these tokens are well-defined. An unquoted token may begin in any of the following positions:

- At the beginning of the file
- Immediately following the end of a quoted token
- Following one or more whitespace characters, outside of all quoted tokens

An unquoted token may begin with any non-whitespace character that does not begin a quoted token. An unquoted token is then terminated by the first of these encountered after the initiating character:

- The end of the file
- Any whitespace character
- The equals sign (ASCII 61)
- Any character that begins a quoted token: `{ < ( ' "`

While the initiating character is considered part of the token, the terminating character may or may not be considered part of the token. Whitespace will be discarded, and quoted token initiators will be part of the next (quoted) token. The equals sign will be kept as part of this token. Note that this means tokens ending in the equals sign (called "Properties"; see 2.3.2) are effectively half-quoted: there need not be whitespace after the equals sign to separate this token from the next (unquoted) token.

The following sections present rules for categorizing unquoted tokens. The rules are in order of precedence—that is, a token must be classified according to the *first* rule it matches from this list. This resolves the ambiguity that would arise if, *e.g.*, a token began with an "at" sign (like a keyword) and ended with an equals sign (like a property).

2.3.1 Keywords

Keywords define the major sections of a kinemage. Each keyword begins with the "at" sign (64). For example, all of the following are keywords:

```
@kinemage @master @vectorlist
```

Furthermore, in order to be recognized as a keyword, the "at" sign must occur at the beginning-of-line (BOL; see 2.1). In addition to enforcing good style, this streamlines the processing of plain text segments (see 2.4).

2.3.2 Properties

Properties are generally used for labeling the meaning of the next token in the file. Each property ends with an equals sign (61). The following are all properties:

```
color= master= radius=
```

Note that, by definition of an unquoted token, whitespace is forbidden before the equal sign. Although some old kinemages may allow this syntax, it requires the tokenizer to read ahead through an arbitrary amount of whitespace following every unquoted token in order to determine if it is a property or not. This behavior could be undesirable if the kinemage contains sections of plain text (see 2.4) or is embedded within some other data format.

As described above, there may be whitespace after the equals sign, but it is not required, even if the next token is unquoted. This semi-quoted (quoted at the end, but not the beginning) behavior of property tokens is a historical feature of the kinemage format that has been retained for backward compatibility. The preferred format for new kinemages is to have a space following the equals sign.

There are no low-level syntactic restrictions on the positioning of properties; however, at a higher level, syntax generally requires that each property be followed by a non-keyword, non-property token. For example:

```
color= red
width= 7
master={backbone}
radius=2.5
```

2.3.3 Integers

Integers are exactly that: text representations of integer numbers. Legal integers are either the single digit zero, or a non-zero digit followed by zero or more additional digits and optionally preceded by a plus or minus sign. The following are legal integers:

```
0 +1 7 -365 2020
```

The following are *not* legal integers:

```
-0 007 5+2
```

Tokens that are not legal integers but consist only of digits 0-9 and the plus and minus signs (*e.g.*, the above) may be interpretted as integers or as literals on a case-by-case basis, at the discretion of the tokenizer. It is recommended that a warning be issued if such a token is encountered.

2.3.4 Numbers

Numbers are a superset of the integers: text representations of real numbers in decimal or scientific notation. Legal numbers follow the pattern[2] below:

```
number ::= integer fraction? exponent?
fraction ::= '.' digit+
exponent ::= ('e' | 'E') integer
```

Basically, there must be something before the decimal point, even if it's a zero; there must be something after the decimal point, if there is one; and the exponential part (if present) may be indicated with either a capital or a lowercase E. The following are legal numbers:

```
-0.42 1e5 3.14 6.022E+23
```

Tokens that are not legal numbers but consist only of digits 0-9, the letters **e** and **E**, the decimal point, and the plus and minus signs may be interpretted as numbers or as literals on a case-by-case basis, at the discretion of the tokenizer. It is recommended that a warning be issued if such a token is encountered.

2.3.5 Literals

Legal unquoted tokens that cannot be otherwise classified are lumped together as literals. Note that, by the definitions provided for unquoted tokens, a literal may begin with a numeric digit. This is in contrast to many programming languages. Those defining new semantics for kinemages are strongly advised against defining literals that are not numbers but use only characters allowed in numbers; the interpretation of such tokens is poorly defined (2.3.4). In fact, it is recommended that literals contain only alphanumeric characters and that they start with a letter rather than a number. The following are all legal literals:

```
animate 2animate red blue green big_long_literal
```

2.4 Plain text blocks

In addition to the ordinary, tokenizable parts of a kinemage file, sections of text data that do not conform to the rules for tokens may be embedded. This data cannot be processed as usual by the tokenizer for two reasons:

[2] See `http://www.garshol.priv.no/download/text/bnf.html` for an introduction to Extended Backus-Naur Form.

1. The data is in an unknown format, and whitespace may be significant.
2. The data may "open" a quoted string but never close it, thereby hiding the remaining content of the file.

An example of this is the plain text write-ups that follow the **@text** keyword; however, future kinemages could conceivably contain embedded HTML, base-64 encoded binary resources, *etc.* At the moment, there is no purely syntactic means for identifying such regions. However, upon the request of the parser, the tokenizer must be able to deliver the unaltered text content of the file from the current position until reaching a kinemage-format keyword (*i.e.*, a new line or carriage return followed by an "at" symbol).

Part II
Semantics: core features

3 Overview of a kinemage file

4 Meta-data

5 Views and display options

6 Masters, aspects, and colors

7 Groups and subgroups

8 Lists

9 Points

Part III
Semantics: selected extensions