XAWK Language Reference Manual

John Cieslewicz
johnc@cs.columbia.edu

Gabriela Cretu
geretu@cs.columbia.edu

Shi Tak Man
sm2173@columbia.edu

Prashant Puri
pp2119@columbia.edu

October 21, 2004
1 Lexical Conventions

This section covers the *XAWK* lexical conventions including comments, identifiers, types of variables and constants, and operators.

1.1 Comments

A single-line comment starts with the character ‘#’ and terminates with newline characters ‘\r’ or ‘\n’. The ‘#’ character does not indicate a comment when it appears within a string literal.

1.2 Identifiers

An identifier, such as a name or a label, starts with a letter followed by letters, digits, or an underscore character ‘_’. Identifiers are case-sensitive. For example, John5 is not the same as john5.

1.3 Keywords

The identifiers listed below are reserved as keywords:

```
break  continue  if    else    text
do     while     for    in     doc
print  printf    true   false
```

1.4 System Variables

There are also built-in system variables besides the keywords. Each of these variables associates with a particular meaning in *XAWK*:

```
ARGC  number of command-line arguments
ARGV  associative array of command-line arguments
PATH  fully-qualified path to current object
CA    associative array of attributes on current element
CE    name of current element
CV    value of current object
      (e.g. text, attribute value, or element name)
```

1.5 Numbers

The two types of numbers in *XAWK* are double constants and integers. Each type is determined by its form and value.
1.5.1 Integers

An integer consists of a sequence of digits.

1.5.2 Double Constants

A double constant, or DoubleConst, consists of an integer followed by two kinds of situations:
- A decimal point ‘.’ followed by an optional exponent and a signed integer
- An exponent followed by a signed integer

1.6 Characters

A character is enclosed in single quotation marks, including the blank character ‘ ’ and newline character ‘\n’. In fact, the new line character belongs to the following set of special characters, which are represented by the escape sequences:

\n  newline
\r  carriage return
\t  horizontal tab
\b  backspace
\f  form feed
\\  backslash
:\'  single quote
:\"  double quote

1.6.1 Unicode escape sequence

In addition to the regular escape sequences, XAWK also recognizes all of the Unicode escape sequences. The following list shows some examples of Unicode escape sequences in the form of \uxxxx:

\u000A  newline
\u000D  carriage return
\u0009  horizontal tab
\u0008  backspace
\u000C  form feed
\\\  backslash
:\'\'  single quote
:\"\"  double quote
\bbb  octal character, where b is between 0 and 7 inclusive and \bbb does not exceed \377
1.7 String Literals

A string literal is a sequence of characters enclosed in double quotation marks. Identifiers are disregarded when placing within string literals while escape sequences are still effective.
2 Expressions

This section describes the forms of XAWK expressions and pattern expressions.

2.1 Forms

An expression can be represented by any one of the following forms:

- `identifier` variable reference
- `identifier [ expression ]` associative array reference
- `@expression` field reference
- `string literal` string literal
- `number` number

2.2 Operators

An operator signifies an operation and can be one of the following:

- `++ --` increment / decrement
  (both pre- and postfix)
- `+ - !` unary plus / minus / logical NOT
- `* / %` multiplication / division / modulus
- `+ -` addition / subtraction
- `< <= > >= != ==` relational operators
- `in` string concatenation
- `&&` array membership
- `||` logical AND
- `?:` logical OR
- `= += -= * = /= %=` conditional (ternary)
- `= += -= * = /= %=` assignment

Left-To-Right associativity is maintained in all of the defined operations in XAWK.
2.3 Pattern Expressions

*XAWK* is a pattern-matching program. Pattern matching commands are executed once for each line in the data file. The order in which patterns are executed varies depending on the object being referenced. For elements and text blocks, actions are executed in the order in which these objects appear in the XML file or tree. Because the order in which attributes appear in XML files is irrelevant, patterns that refer to attributes are executed in the order in which they appear in the source file.

2.3.1 XPath syntax

A subset of XPath syntax is displayed below:

- `doc(expression)` matches a file or tree
- `identifier` matches a child element with the given name
- `*` matches any child element (not attributes or text)
- `text()` matches text children
- `@name` matches a given attribute
- `@*` matches all attributes
- `/path1 /path2` matches children of `path1` that match `path2`
- `/path1 //path2` matches any nodes below `path1` that match `path2`

The XML specification requires that tag names start with an upper- or lowercase letter followed by letters and numbers as well as the period, dash, and underscore characters. A colon is also common in XML tag names so *XAWK* allows that character as well. The colon, period, and dash are not allowed in standard identifiers, so tag names including these characters must be enclosed in single quotes. For example:

```
/’path’/path
/path//’path’/
```

The following will be correctly recognized as valid paths:

```
/ class / student
/’stu-dent’ / grade_6
/
/ @*
/ student // grade
```

The expression of the `doc` directive will be considered as a file path name when expressed as a string.
3 Statements

This section describes the forms of XAWK statements.

3.1 Forms

A statement can be represented by any one of the following forms:

```
break [ label ] ;
continue [ label ] ;
do statement while (expression) ;
extression ;
if ( expression ) statement [ else statement ] ;
for ( expression ; expression ; expression ) statement ;
for ( expression ) statement ;
I/O statement ;
for ( variable in array ) statement ;
while ( expression ) statement ;
pattern { statements }
{ statements }
;
```

Any statement may be preceded by a label:

```
label :
```

3.1 Examples

Forms of statements are given below:

3.1.1 Assignment statements

```
identifier ( index ) = expression
identifier ( index ) += expression
identifier ( index ) -= expression
identifier ( index ) *= expression
identifier ( index ) /= expression
```

The ( index ) part is optional in the assignment statements.

3.1.2 Label statements

```
identifier : statement
```
3.1.3 Iterative statements

Iterative statements are simply loops. There are two kinds of loops in XAWK, for loops and while loops.

**do** statement ---

```
do statement while ( expression )
```

**while** statement ---

```
while ( expression ) statement
```

**for** statements ---

```
for ( for-condition ) statement
```

*for-condition* is defined as follows:

```
expression ; expression ; expression
```

```
for ( expression ) statement
```

*expression* has to be an expression of the form: variable in array

**break** statement ---

```
break ( identifier )
```

**continue** statement ---

```
continue ( identifier )
```

3.1.4 Conditional statements

```
if ( expression )
```

```
statement
```

**else**

```
statement
```

The **else** block is optional in the conditional statements.
3.1.5 I/O statements

\[
\text{\textit{print}}(\textit{io})
\]

or

\[
\text{\textit{printf}}(\textit{identifier io})
\]
\[
\text{\textit{printf}}(\textit{stringliteral io})
\]
\[
\text{\textit{printf}}(\textit{( expression ) io})
\]

where \textit{io} is defined as follows:

\[
\begin{align*}
\text{stringliteral} & \quad + \quad \text{stringliteral} \\
\text{identifier} & \quad + \quad \text{identifier} \\
\text{identifier ( index )} & \quad + \quad \text{identifier ( index )}
\end{align*}
\]

where the \{ \text{stringliteral / identifier / identifier ( index )} \} is optional.

3.1.6 Pattern statements

\[
\text{pattern statement}
\]

where pattern is in one of the forms defined in pattern expressions in section 2.3.