THE HAWKX PROGRAMMING LANGUAGE
VERSION 1.0

Gregory M. Baumgardner
gmb2108@columbia.edu
COMS W4115 Fall 2005
Prof. Stephen Edwards
February 8, 2005

1 Introduction

1.1 Overview

The ***hawkx*** programming language is designed to parse and manipulate simple textual data, and build up an XML representation of the data. First, by offering *awk*-like features and constructs, this new language uses pattern matching on the text, splits input lines into separate fields, and offers a wide array of functions that can manipulate the data in the fields. Likewise, the text can be built into an XML data structure using the W3C standard XPath syntax.

1.2 Background

The simple text file has been embedded in configuration files and logs since the early computers. In particular, the UNIX operating system and its variants almost exclusively use simple text format files to setup up everything from user and preference information to network and security configurations. Likewise, many applications running on the operating system might persist certain data or write log files in readable text. In almost all of these cases, the so-called "flat-file" was well-formed. That is, the structure of the file was fixed and had well understood meaning. It was this fact that led to the development of a general purpose parsing tool for text files, simply known as *awk*.

Named for its creators – AT&T Bell Lab's Alfred V. ***A***ho, Peter J. ***W***einberger, and Brian W. ***K***ernighan – *awk* quickly gained popularity for its power and ease of use. The tool automatically parses input line by line, and allows the programmer to selectively alter in various ways. Over the years, numerous systems programmers found it more than convenient to develop *awk* scripts to extract or modify data from all sorts of UNIX configuration files. Even today, *awk* is usually language of choice to quickly parse and manipulate formatted text files.

With the advent of the world wide web, markup languages such as HTML (Hypertext markup language) and its later, more general cousin XML (Extensible Markup Language) began to dominate the text formatting for computer files. Fortunately, standards bodies such as the W3C (World Wide Web Consortium) designed many aspects of theses languages to enable more productive tools for file manipulation and application data exchange. The Xpath notation provides a rudimentary description of generic locations within an XML document. XML Schemas and Document Type Definitions (DTD) provide a couple of optional languages for describing the meaning of the markup tags within an XML document. The XML Style Sheet Language Transforms (XSLT) is a W3C language recommendation that enables conversions from one XML grammar to another. The Document Object Model (DOM) is a powerful object data structure representation of XML files that allow for computing differences between files. Simple Object Access Protocol (SOAP) provides a channel to transfer objects between applications in a textual XML representation. There are many other examples.

It appears clear that the markup languages, particularly XML, present a powerful yet comprehensible framework for textual manipulation. However, since there still exists a significant presence of regular flat files on every computer system, there appears to be a need for some language that can help quickly markup text files in order to be able to better perform the complicated tasks that are natural to XML documents. This new language is known as ***hawkx***, a cross between the ever popular *awk* language for text parsing and Xpath notation for building the XML document.

1.3 Motivation

The ***hawkx*** language is designed to solve a fundamental problem with text file manipulation. Although simple programs in *awk* or similar languages exist for changing contents in a file or appending new data, it has proven extremely difficult to merge similar content from two or more files. For example, suppose a system administrator was consolidating two old servers into a single advanced system. Each old server hosted hundreds of user groups with dozens of users in each. Some of the groups may overlap, and some of the users may overlap between the two old servers. The task for the administrator is to merge all the thousands of users and hundreds of groups onto the new machine. Appropriately handling the duplicates clearly is a daunting task for a script to manage. Even more complicated may be that user group listings have some known character limit within the text file that requires entries to be split into more than one line. This kind of problem demonstrates the motivation behind the ***hawkx*** programming language. Instead of creating a complicated script with all the logic necessary for merging the user and group files, the system administrator defers those functions to a higher level language with a more sophisticated utility set such as Java. Instead, the scripting required is solely to provide useful input into the Java code. Clearly, using XML files would allow the programmer to merge the content of those files quickly and effectively, and perhaps an XSL transform can be used to write out the resultant merged XML file back to the UNIX configuration file format. Therefore, it is the primary objective of ***hawkx*** to provide a language that can naturally parse the text files and construct an XML representation with little effort.

1.4 Goals

Document Construction

The primary goal of ***hawkx*** is quick and natural XML document construction. The syntax is designed to be a flat, directory-like structure for the document being constructed. File pointer references are used to track the current location within the document, and data may be pushed into or pulled from a location relative to a file pointer, or from the absolute root of the document. The markup tags within the XML document are created by simply stating their location along the path. No intermediate data structures are required.

Automatic Parsing

Following in the footsteps of *awk*, this new language will read input files one line at a time, and will parse each into field, accessible through positional parameters. The default white space field separators may be overridden in order to configure the parser to recognize different formats of input lines. The developer may also reset the positional parameters within the program.

Portability

Even though flat text files are prominent on UNIX systems, they exist on all platforms. Thus, the ***hawkx*** interpreter should be portable to any system. Most importantly, a program that runs on one platform should generally run the same on another. This does not necessarily, however, take into account the uniqueness of a specific file or operating system intentionally exploited by the programmer.

Error Handling

An important goal of the language is to detect or prevent errors in the XML document. First, the interpreter is obligated to produce well-formed XML, per the XML 1.0 standards. There shall be no direct manipulation of the elements or attributes allowed that would jeopardize this guarantee. Secondly, all encoding shall be UTF-8 to reduce the possibility of invalid characters being passed into the document. Finally, the programmer may optionally supply a data definition in the form of XML schema or Document Type Definition (DTD). If supplied, the XML document shall conform to language defined, or a run-time error will occur. The intent here is to better ensure that the document constructed will be accepted by any higher level application written to manipulate the input.

1.5 Features

Simple Data Types

Since the XML document exclusively contains textual strings, the string is the primary data type in ***hawkx***. All simple variables are of string type, although strings containing only digits may be accessed as a numerical value within an expression. Literal strings should be enclosed in double quotes (“”) with standard XML entity encoding for special characters. The only complex data type allowed are document node pointers, which tracks the path along the XML document. Because an XML document represents a complex data structure, it can serve as pseudo-array data structures, but unless the data is meant to be saved in the document, it needs to be removed after use. No variable ever needs declared in the program. Instead, the type is determined by the way it is accessed, with the dollar sign ($) for string expansion and ampersand (&) for document position dereferencing. Finally, logical expression evaluate to boolean types which apply to some language constructs, but can also be used as a string.

Variables

All variables in a ***hawkx*** program are considered global. This is in order to maintain the state of a variable throughout each iteration over the input. Obviously, a variable can be reset at any point within each iteration, which effectively behaves like a local variable. Several special variables exist in the language. The positional parameters ($1, $2, ... $NF) represent the fields parsed automatically by the interpreter or through a re-split function. The $NR variable is the ordinal number of current record from the start of the input. The &CONTEXT variable represents the default document reference. Variables can be set through the assignment operator (=).

Arithmetic Operations

When used within expressions, strings containing numerical values also support arithmetic operations including addition (+), subtraction (-), multiplication (*), division (/) and modulus (%). These operations are done with double floating point precision. Comparison operations include equal to (==) , not equal to (!=), greater than (>), less than (<), greater than or equal to (>=) and less than or equal to (<=). Each evaluate to a boolean.

Pattern Matching

Any string type in ***hawkx*** can be compared to a pattern enclosed in pipes (||) using the matching (~) or not matching (!~) operators. The result is a boolean type to be used in logical expressions. Some regular expressions are supported in the pattern.

Built-in Functions

Several functions are predefined for use in the language. These include all of the standard XPath functions, including those that accept or return a document position. Additionally, the resplit() function accesses the automatic parser to reset the positional parameters.

XML Document Construction

The XML document is constructed by accessing path from document node pointer. This is done through the dereferencing operator (->), which is assumed when not explicitly using the &CONTEXT variable. Full XPath notation is supported for describing the document path, including both the full and abbreviated syntax. Textual data may be pushed into the document path with the insertion operator (<<), and can be pulled out from the document under construction with the extraction operator (>>). The latter operation is necessary to assign data stored in the XML document to some string variable.

XML Schema/DTD Support

The ***hawkx*** language automatically generates well-formed XML, but can also be configured to validate the document structure against a known XML schema or Document Type Definition (DTD). Any nonconformance will generate a fatal error in the program.

1.6 Sample

```
# Comments begin with a pound sign
# and go to the end of the line

# This program builds an XML from /etc/passwd

if ( $1 ~ |^#| ) then { next }  # Skip comments
resplit( |:|, $0 )              # Split the colon-separate line

&/passwd/user <<= $1  # Push first field into user node under /passwd
                      # Move the pointer to the newly created node

&ref = &CONTEXT        # Save off a reference to context node location

&attribute::uid << $3  # Create an attribute in the current node
                       # Relative path is assumed from &CONTEXT->
                       # Do not move the context node pointer

$comment = $5       # Save off the text for later use

&ref->home << $6    # Relative path from reference variable
&ref->shell << $7   # Do not move the ref pointer

resplit( |,|, $comment )   # Break apart comment field by commas
&ref->name/last << $1
&ref->name/first << $2

if ( $3 !~ |^$| ) then { &ref->office/room << $3 }
if ( $4 !~ |^$| ) then { &ref->office/phone << $4 }

END {            # Run after all lines have been processed
  $usercount = count(&/user)  # Count the number of users
  &/[1] << comment("There are " + $usercount + " users")  # Create an
                                                          # XML comment
}
```

When this program is run against the following /etc/passwd example:

```
# Sample /etc/passwd file
johndoe:x:345:10:Doe, John, Room 405, x2765:/home/johndoe:/bin/ksh
noname:x:346:10:Noname, Mr.:/usr/home/noname:/bin/bash
```

It produces the XML document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- There are 2 users -->
<passwd>
  <user uid="345">
      johndoe
    <home>/home/johndoe</home>
    <shell>/bin/ksh</shell>
    <name>
        <last>Doe</last>
        <first>John</first>
    </name>
    <office>
        <room>Room 405</room>
        <phone>x2765</phone>
    </office>
  </user>
  <user uid="346">
      noname
    <home>/user/home/noname</home>
    <shell>/bin/bash</shell>
    <name>
        <last>Noname</last>
        <first>Mr.</first>
    </name>
  </user>
</passwd>
```

1.7 Summary

With the aid of ***hawkx***, simple text files can be marked up with a valid XML grammar. Instead of filling the language with a bunch of bells and whistles that allow for complex data structures to be built or powerful calculations to execute efficiently, this simple language is built solely for the purpose of processing strings and using them to build XML documents. The document path syntax follows that of the W3C XPath specification in order to ease the XML construction. Blending the capabilities which made the *awk* language popular with the flexible nature of XML, the ***hawkx*** language is sure to become the bridge between legacy text and enhanced document markup.