Programming Languages and Translators

COMS W4115

Department of Computer Science

Fall 2007

TweaXML

Language Proposal

Kaushal Kumar
kk2457@columbia.edu

Srinivasa Valluripalli
sv2232@columbia.edu

Abstract

In this document, we present the proposal for TweaXML, a high level language for manipulation of XML documents. We will discuss the motivation behind the idea and key features of the proposed language. Also, we will provide a brief syntax and example in this language.

Introduction

TweaXML is a simple language that helps us manipulate and perform various operations on XML documents, like extracting specific data, performing arithmetic/string operations on the extracted data and saving the data in a desired format to a file.

The Extensible Markup Language (XML) is a general-purpose markup language. It is classified as an extensible language because it allows its users to define their own tags. XML is designed primarily to share information across heterogeneous systems, regardless of the specific architecture or language of the systems. It is adopted as an interoperable language across various technologies and component vendors, for example, it is used to share data across J2EE components and .NET components.

Since XML is highly customizable and contains user-defined tags, the technologies to parse an XML document are quite complex. For example, Java provides APIs, like SAX or DOM Parsers to parse XML documents, but they are quite complex and require a thorough knowledge of Java. Therefore we propose to create TweaXML, a language which will be easy to use and will require minimal amount of apriori knowledge of programming to get started right away.

TweaXML will enable a novice user to manipulate and operate on XML documents, extract its data, and do arithmetic/string operations on it and save it in a desired format. TweaXML will be written in Java and it will use APIs provided by Java, like SAX and DOM parsers, to handle the XML documents. We plan to provide facilities of user-defined functions, basic arithmetic/string operations, writing data to a file etc.

Language Features

Data Types

NUMBER – Generic date type for any kind of number, integer, float or double.
STRING – Data type containing data of character and string types.
FILE – Data type for reading/writing file on a file system.
NODE – Data type for a XML element.
BOOLEAN – Standard true/false Boolean type.

Operators

Arithmetic Operators:

+	(addition)
-	(subtraction)
*	(multiplication)
/	(division)
=	(assignment)
==	(equal)
>	(greater than)
<	(less than)
>=	(greater than or equal to)
<=	(less than or equal to)
!=	(not equal to)

Scope Operators:

"{" , "}"
"(" , ")"
// comment

Looping constructs:

If() {…} else {…}
While(){…}
FOREACH(){…}

User-Defined Functions:

User can define functions and call them from another one. The code starts executing from a function “main ()”.

Methods available on STRING, NODE and FILE data-types:

STRING:

- Standard methods of java “java.lang.String” class.

NODE:

- countChildren() – returns number of nodes inside the node.
- containsNode() – returns true/false.
- getValue() – returns the value of the node.
- getChild(“xxx”) – returns child node of the node with name xxx.

FILE:

- open() – opens a file to read/write.
- read() – reads a file.
- close() – closes a file after processing.
- write() – writes in the file.

Sample Program:

Input file: (input.xml)

```
<customer-records>
	<customer>
		<name>John</name>
		<order-number>1</order-number>
		<amount>100</amount>
	</customer>
	<customer>
		<name>Jack</name>
		<order-number>2</order-number>
		<amount>50</amount>
	</customer>
	<customer>
		<name>Harry</name>
		<order-number>3</order-number>
		<amount>150</amount>
	</customer>
</customer-records>
```

Program:

```
main(){
	// declaring FILE variable.
	FILE input;
	FILE output;

	// opening the file to read
	input.open("input.xml");
	// opening a file to write in a comma separated format.
	output.open("output.csv", ",");

	// gives the root node of the xml document.
	NODE rootNode = input.read();

	// variable to calculate the total amount.
	NUMBER totalAmount = 0;

	if(rootNode.contains("customer-records/customer/amount"))
	{
		// looping over all the nodes with name “customer-records/customer/amount”
		FOREACH(NODE nextNode = rootNode.getChild("customer-records/customer"))
```

```
                {
                        output.write(nextNode.getChild("name").getValue(),
                        nextNode.getChild("order-number").getValue(),
                        nextNode.getChild("amount").getValue());
                        output.newLine();
                        totalAmount = totalAmount + nextNode.getChild("amount").getValue();
                }
                output.write("Total Amount: ", totalAmount);
        }
}
```

This program will read the input file (given above), and outputs the data in a comma separated csv file. Additionally, it will add the values of all the “amount” nodes and print the added value at the last line of the output file.